AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Bachelor Project
A digital microscopy system to classify sediment samples of foraminifera

by

Daniel Christopher Biørrith, 201909298

Lukas Koch Vindbjerg, 201906015

Bachelor's Thesis

in

Computer Engineering

Supervisor: Kim Bjerge

Aarhus University

Department of Electrical and Computer Engineering

June 15, 2022

Character count: 69.488

Chapter 1

Abstract

Foraminifera are small, single-celled organisms that are used extensively in research in a wide range of areas from Earth's climate history to oil deposit analysis. The process of identifying and analysing these organisms is typically done manually and can be quite an extensive endeavour. The purpose of this project is to develop an automated system capable of automating the said process.
The system consists of three major steps; acquisition, segmentation, and classification. The acquisition step uses a digital microscope mounted to a CNC router in order to traverse and capture images of a sediment sample on a tray. The segmentation step uses digital image processing methods to stitch together these images and isolate individual objects. Finally, the classification step uses a convolutional neural network to classify these objects into groups of either sand or foraminifera. The result of the project shows that an automated process is achievable with the system. However, there is much room for advances in the design and future work.

Resume

Foraminiferer er små encellede organismer, som bruges i vid udstrækning i forskning på en lang række områder i alt fra Jordens klimahistorie til analyse af olieaflejringer. Processen i at identificere og analysere disse organismer udføres typisk manuelt og kan være en omfattende process. Formålet med dette projekt er at udvikle et system, der er i stand til at automatisere denne proces.
Systemet består af tre hovedtrin; anskaffelse, segmentering og klassificering. Anskaffelses trinnet bruger et digitalt mikroskop monteret på en CNC fræser for at skanne over, og tage billeder af, en sedimentprøve på en bakke. Segmenteringstrinnet bruger digitale billedbehandlingsmetoder til at sammensætte disse billeder og isolere individuelle objekter. Til sidst bruger klassificeringstrinnet et convolutional neural network til at klassificere disse objekter i grupper af enten sand eller foraminifer. Resultatet af udviklingen viser, at en automatiseret process er opnåelig med dette system. Der er dog meget plads til forbedringer i designet og det fremtidige arbejde.

Chapter 2

Preface

This bachelor project has been written by Daniel Christopher Biørrith and Lukas Koch Vindbjerg at the department of Electrical and Computer Engineering at Aarhus University. The projects scope is 15 ECTS for each student where the objective and learning outcome have been specified by our supervisor, Kim Bjerge. We chose the project based on a mutual interest in fields such as digital image processing and deep learning, which the project would cover. Furthermore, we found it inspiring to develop and design something that new, which was proof-of-concept to test if the concept was possible.

We are among the first students enrolled in this education of Bachelor in Computer Engineering at Aarhus University and are therefore among the first to complete the corresponding bachelor project. We have worked in close collaboration with our supervisor throughout the project to develop and tailor the project to fit our learning goals. We are grateful and would therefore like to thank Kim Bjerge for his support and participation in this project.

By
Daniel Christopher Biørrith and Lukas Koch Vindbjerg

	Daniel	Lukas
Machine Learning	P	
Segmentation		P
Stitching		P
Acquistion implementation	P	S
Acquisition research	P	P
Documentation	P	P

Table 2.1: List of Contribution. P means primary while S means secondary.

Glossaries

Acronym and technical terms	**Description**
DFI	Digital Foraminifera Identifier, the formal name of the project, which it will be referred to as throughout the report.
CNC system	Computer numerical control system.
Pylon 6	A software provided by Basler. It comes both with a program, Pylon Viewer, which makes it possible to get livefeed from the microscope, as well as an SDK for developing code to control the camera.
SDK	Software development kit
Foraminifera	A singlecelled organism often found in sediment samples.
Sediment sample	Sediment samples are samples consisting of foraminifera and other granular material.
Tray	The observation platform used to place sediment samples on during scanning.
G-code	A highly popular CNC programming language, used to communicate with the controller.
Grbl	The software located on the controller, which interprets the G-code sent.
VS2013	Visual Studio 2013 is an IDE provided by Microsoft, which is used to develop the acquisition software.
DIP	Digital Image Processing
CV	Computer Vision
GUI	Graphical user interface.
ML	Machine Learning
USB	Universal serial bus
OpenCV	An open source CV library for Python.
Scikit-image	A collection of algorithms for image processing, developed for Python.
Tensorflow	An open source CV library in Python, used for creating ConvNets.
JupyterLab	An IDE primarily used to execute .ipynb files.

Table 2.2: List of glossaries and descriptions thereof

Contents

Chapter 3

Introduction

3.1 Background

Foraminifera are single-celled organisms as small as 100 micrometres, abundant in the fossil record for the last 540 million years. They are both quite abundant in sediment samples and are extremely sensitive to the environment and chemistry in which they grew. For this reason, they are an invaluable asset in many research areas, from exploring the historical timeline of the earth, to locating oil deposits[1]. As research involving foraminifera is so effective and extensive, it is inevitable that an vast amount of samples have to be collected. Done manually, this task can become quite labor intensive, as it requires both an adequate knowledge of the subject alongside precision and focus.

3.2 Objective and purpose

With this project, formally known as Digital Foraminifera Identifier (DFI), the intend is to approach the issue by trying to automate the process of taking a sediment sample and detecting foraminifera. Consequently, a system is wanted, where a user can provide a sample and then, with no further manual work, the system will end up with a set of images of the foraminifera. The way it is thought to be achieve can be separated into three major components: acquisition, segmentation and classification. The acquisition component consists of the process of getting from the physical sediment sample to a structured digital framework of images. The segmentation component consists of converting the previously mentioned images into a set of individually isolated objects, where it ensures a consistent format. Finally, the classification component will go through each element and classify whether it is foraminifera or sand.

The scope of this project is proof-of-concept. The project will therefore solely focus on the three components mentioned above. The expected outcome is an exploration of the theories, design decisions, and adequate testing and results. Aspects of a complete system, such as a user friendly UI or a sophisticated storage architecture, will not be covered or developed in this project. The aim is to test the possibilities and capabilities of the system proposed above.

As such, it is worth exploring previous research regarding this topic. Notably, two previous projects at Aarhus University have worked on this issue; ForaminiferaScanner and FRIDA[2][3]. This project is a continuation of those, further developing on their proposed solutions. Furthermore, external papers and material have worked on areas similar to the one in this paper. However, these have mostly focused on narrower aspects and not tried to integrate a system capable of handling the entire process[4][5].

3.3 System description

This section will provide a more detailed description of the system, its components, and which components are designed to handle which actions. The setup of the system can be seen in figure 3.1.

The idea is to use a digital microscope to capture images of the sediment sample. The sample should be spread out, such that none of the granular material obscure one another. For this, the black tray, shown on figure 3.1, is used as a consistent platform for the samples. The microscope only captures a very small area of the tray and can not capture more than a few dozen grains in one image. Consequently, it has to be moved across the whole tray to capture the entire sample. To do this, the previous project, ForaminiferaScanner, has rigged the microscope to a CNC router. This allows for the possibility to control the movement of the microscope to traverse any specified area, with periodic stops, such that the microscope can capture clean

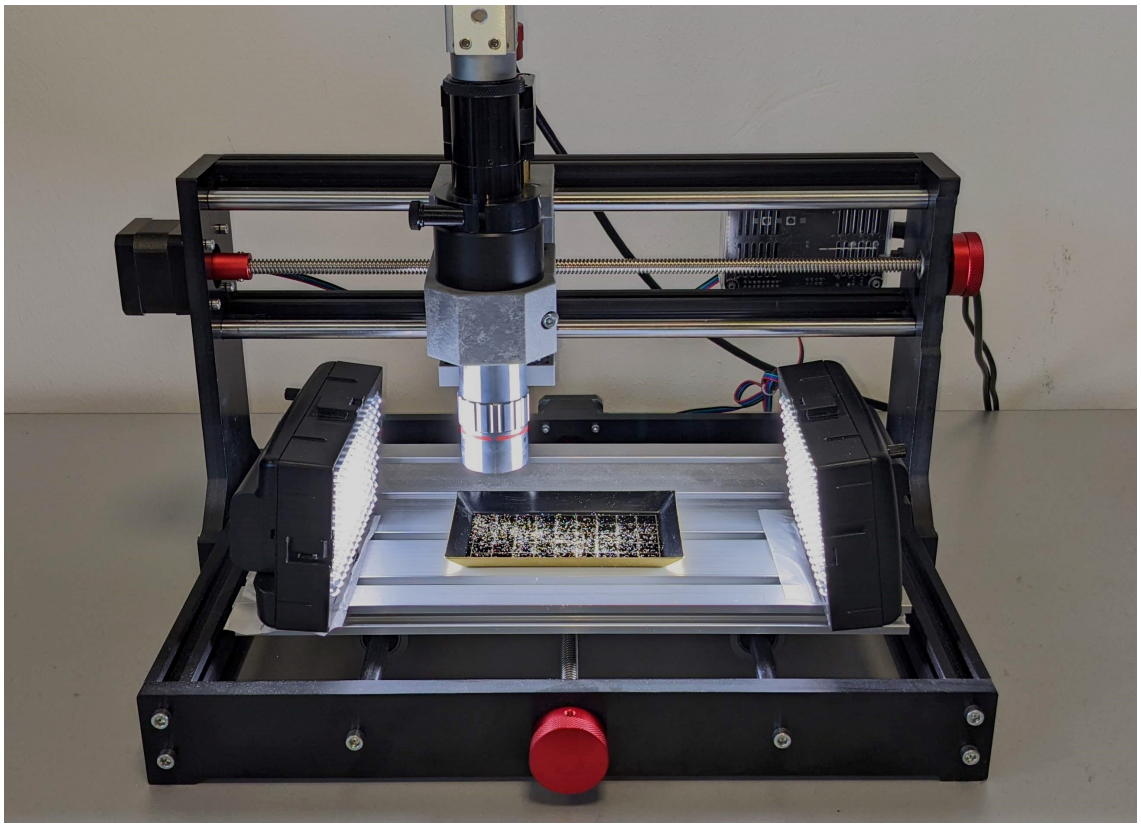


Figure 3.1: Setup of the DFI system

images. The CNC router is controlled by a control board attached to the setup. Both the microscope and the control board can be controlled from the user's PC. This allows the user to change settings and procedures depending on their intended use. Finally, a pair of LED lights have been mounted to the system to provide the sensor of the microscope with sufficient lighting, to help optimize the process and maximize the information preserved from the real world sample. With the setup initialised, the acquisition process can be started and the raw images will be stored on the user's PC.
With all the images acquired, the next process of the system is to isolate individual objects. This will be done using theory and methods from the field of digital image processing (DIP). The selected images will first be stitched together into one large image and then the granular material will be segmented into images which only contain a single object. These will again be stored on the user's PC.
Finally the system will classify these segmented objects using a convolutional neural network (ConvNet). It will classify an object as either foraminifera or sand. These sets of both foraminifera and sand will likewise be stored on the user's PC in separate directories.

Chapter 4

Process

This chapter will explore the process which was followed to realize the purpose of the system. As the project covers a wide range of disciplines, a structured approach will help to ensure a stable development process.

Since this project will contain development of essentially isolated units - acquisition, segmentation and classification - to fulfill the final system, an approach that allowed for separated development of different units with later integration would be desirable. Thus, it was decided that this project would follow the waterfall model[6, p. 45]. This model allows for planing of all the processes before development. The waterfall model followed consists of the following phases:

1. Requirements and definitions. This step will be the foundation for the rest of the development process, as it specifies which system units should be designed.

2. System design. Here each individual aspect of the system will be designed to meet the specified requirements.

3. Implementation and testing. In this stage each designed unit will be implemented and individually tested to ensure that they meet the specified requirements.

4. Integration and system testing. Here the different units will be combined to test the complete system. The processes of the DFI system will execute sequentially, which means that each stage of the process will correctly satisfy its requirements so that the next stage can begin correctly.

Depending on the system in question, the tolerance for overlap of these phases can differ. As this project will deal with issues with no obvious design direction, it will be advantageous to allow for flexibility in decisions. Newly acquired knowledge can be applied retroactively to optimise the system without compromising the overall structure. Another deviation from the waterfall model is that the initial part of the development process was devoted to research and testing of capabilities. As this project extends on the products of the previous projects, it was decided to first explore the limitations and possibilities of the provided hardware. This allowed for more refined requirements and definitions initially.

After the initial research, the rest of the model was followed as specified in the phases above.

Chapter 5

Requirement specification

5.1 Introduction

To help realize the purpose of DFI, a requirement document was created. The document provides a description of the functional and non-functional requirements, which in turn provides a description of what the system should do and how it should perform. Additionally, based on said requirements, a test specification was included in the document. The full document can be found as an appendix[7]; here, a summary of the key point will be presented.

By applying the requirement engineering process[6, p. 101] we have derived the functional- and non-functional requirements of the DFI system. The functional requirements are presented both by some statements regarding how the system should work, and by applying the use-case method[6, p. 125], where each use-case is defined using (i) the natural language specification[6, p. 121] and (ii) the structured specification[6, p. 122]. Furthermore, the non-functional requirements are presented using only the natural language specification.

5.2 Functional Requirements

The following section will present the functional requirements of DFI. Some functionalities that the system must have are described below:

FR01: The motors controls the movement of the microscope.

FR02: The motors must move the microscope systematically over the samples, with slight image overlap, such that it doesn't miss any potential Foraminifera.

FR03: The camera must stand still for a fixed amount of time, depending on the exposure time, providing the microscope enough time to grab an image that is in focus.

FR04: The system must be deployable on a PC, on which Windows 11 is installed.

FR05: The images should be stored on the host PC, so that the user may access and review them manually.

5.2.1 Actors

In order to understand the use-cases, we first have to define the actors using the system, as they are actuating the use-cases. Name, type and description of each actor can be found in table 5.1.

Actor name	Actor type	Actor description
User	Primary	The manual user of the system. The user is the only primary actor, meaning it's the only actor responsible for starting the different parts of the system.
Sample	Secondary	The sample that is to be analyzed, which contains the foraminifera. This should be distributed onto the tray as evenly as possible. The sample is a secondary actor as it is necessary for the primary actor to perform certain use cases.

Table 5.1: Actors of the DFI system

5.2.2 Use case diagram

With the actors defined, a use-case diagram of the system was developed. The diagram is depicted in figure 5.1. In addition to this, the natural language specification/description of each use case is provided in table 5.2.

ID	Name	Description
UC1	Set settings	The actor sets variables, such as exposure time and depth, to their wanted value for optimizing the images. When the preferred values are chosen, the actor can either save the values or cancel, which would prompt to go back to the previously set values, or default in case none are set prior.
UC2	Scan samples	Following UC1, when the actor considers the system and camera to be set up correctly for the desired task, the actor starts the scanning process. This prompts the system to systematically move the camera and capture images of all the samples. These images are then stored in a local directory with an appropriate naming scheme.
UC3	Stitch images	Following UC2, the images needs to be stitched. If the images have been capture, the actor may start the image stitching program with images stored in a local directory on the host machine. After starting the program, it will start stitching all the individual images together into one large canvas. When finished, this canvas will likewise be stored on the local machine.
UC4	Segmentation	The actor provides the segmentation program with an image, in which the actor wishes for the present object to be detected and isolated. Here, the object will be detected using digital image processing methods. These objects will then be isolated into individual images. Finally, the program will process the images of isolated objects in order to get a consistent format.

UC5	Classify images	The actor provides the classification program with a directory in which the image(s) are located. The image(s) must be of an isolated object, possibly following UC4. When the actor starts the program, a trained model determines whether the provided object is a foraminifera or sand.

Table 5.2: Use-cases natural language specification

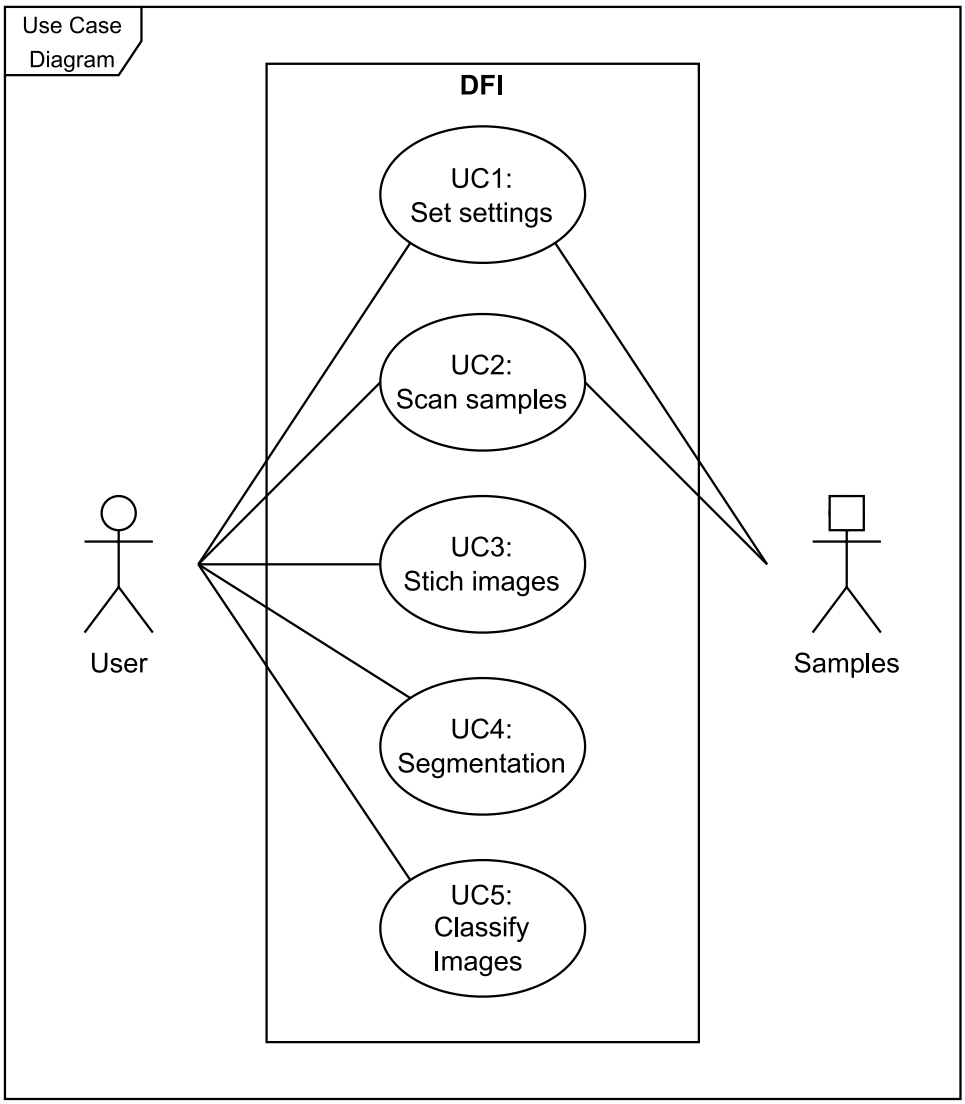


Figure 5.1: UML Use Case Diagram

A fully-fledged description of all use-cases can be found in the requirement document[7].

5.3 Nonfunctional Requirements

Besides the functional requirements, some non-functional requirements were also derived. These help put a constrains on services offered by the system. They are as follows:

NF01 The scanning of the samples must be done within a time-frame of 1 hours 30 minutes.

NF02 The stitching of images must be done within a time-frame of 2 hours.

NF03 The segmentation of an image must be done within a time-frame of 2 hours.

NF04 The classification of the sample images must be done within a time-frame of 2 hours.

NF05 The tray used for the samples must be 9,5 x 5,5 cm in size.

NF06 The classification must have an f1-score of over 85%.

NF07 The communication between the camera and the host computer is through a serial connection.

NF08 The communication between the controller and the host computer is through a serial connection.

5.4 Testing specifications

As mentioned, a testing specification was also made. Each test case will be marked with one of the following:

- **Passed**
  A ✓ will represent passed. It means the test case is accepted.
- **Passed w/ remarks**
  A (✓) will represent passed with remarks. It means the test case is partially accepted but with certain remarks explained in the 'Comments' column.
- **Failed**
  A ÷ will represent failed. It means the test case is not accepted.

A summary of the use case tests will be included in results, in a filled out version of table 5.3.

Use case testing summary		
UC No.	**Results**	**Comments**
UC1		
UC2		
UC3		
UC4		
UC5		

Table 5.3: Summary of use case testing.

Chapter 6

Technology Research

A more thorough look into the available technologies is necessary to realise the desired characteristics determined in the requirement specifications. Therefore, this chapter will go into the background research of the technologies used in the system. This includes the general methods behind the digital acquisition, image processing and machine learning with a focus on getting a background understanding of the topics. Furthermore, an overview of existing technologies and software will be explored. The aim of this is to get a more comprehensive understanding of the possibilities and future design decisions of the system.

6.1 Acquisition

The acquisition part of the DFI system is designed to satisfy usecases 1 & 2. The implementation can be divided into two sections: the capturing of the images and the movement of the camera. This section will explore the hardware, software and the interface between the two. The hardware of the system is primarily the same as the hardware used for the previous project, ForaminiferaScanner[2].

6.1.1 Digital microscopy

The microscope setup consists of multiple hardware components. The camera, which is a Basler acA2440-35uc, connected to a Mitutoyo M Plan APO 5X lens[8][9]. The lens has a working distance of 34 mm with a field of view of 0.96x1.28 mm when using a digital camera with a 1/2" chip size. Furthermore, the lens has a depth of field of only 14 μm[3]. The camera has a resolution of 5 megapixels which gives an image with a size of 2448 px x 2048 px. The sensor size is 8.4x7.1 mm which is a larger format than the one used for reference in the lens' specifications. This results in a larger field of view and each image will therefore cover an area larger than 1 mm in both width and height.

Both the exposure control of the camera sensor and the concrete image capturing and storing from the camera are programmable via the camera API. The camera interfaces through USB 3.0 both for power supply and controllability. Basler has provided a software development kit (SDK) to allow for easy application development[10]. The framework consists of drivers and tools with documentation and sample implementation to illustrate potential uses. The SDK is designed for use on a Windows PC and allows for implementation in both C, C++ and C#. Building an application with the SDK requires either Visual Studio Community or Visual Studio 2013[11].

6.1.2 Computer numerical control system

The movement of the microscope is done with the use of a desktop computer numerical control (CNC) router. It consists of three stepper motors, capable of moving the microscope in the x, y, and z-axis relative to the CNC table. The motors are controlled with a Woodpecker CNC Control Board V3.4. The board is connected to a 24V grounded power supply which is used to power the motors, while the control board itself is powered by the USB connection to a PC. The board is Grbl based which is a motion control firmware designed for CNC machines[12]. It is an embedded, high-performance G-code-parser[13]. G-code itself is a highly popular CNC programming language. The parameters - such as distance units - can be modified to fit a task. Using milimeters as the distance unit, the Grbl execution of G-code allows for values of one decimal place. This gives a possible accuracy of 0.1 mm, if assuming no motor movement offset. The G-code commands that the board processes are sent through the USB of the connected PC.

6.1.3 Considerations

As the microscope can only capture an image of a small area, attempting to capture images of the entire tray will lead to some complications. The more images captured, the more image edges, which might cut the granular material in half . As the system can not use an image of half a foraminifera, it is essential that a complete image of each object is obtained. This is a partial reason behind the stitching requirement.

Although stepper motors are typically considered quite reliable and precise, this system requires exceptional accuracy, due to the amplified effects on the microscope. Moving the camera such that the bottom of one image perfectly matches up with the top of the image directly below can not be consistently relied on. This becomes especially apparent by looking at figure 6.1. Here, the two blue circles show how the same object does not line up from one image to the next. After the top image was captured, the microscope was moved 80 mm to the right, 1 mm down and then 80 mm back again, which should result in vertical alignment.

This test further underlines the importance of the requirement for stitching. Without stitching, the system would not produce a reliable result. Furthermore, as overlap is required, object may appear in multiple images, skewing the result as duplicate entries would appear.

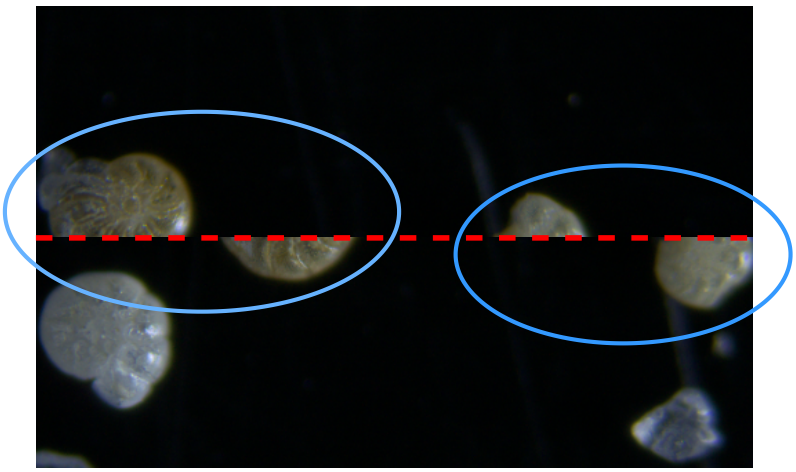
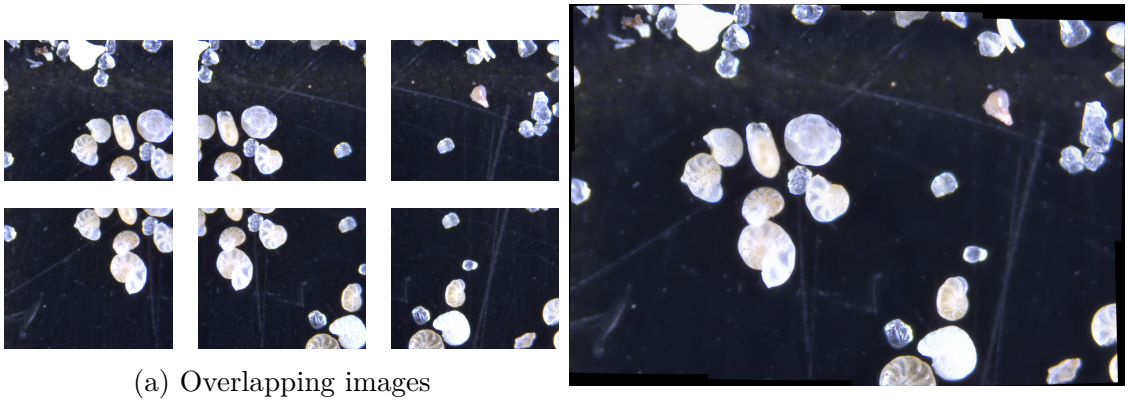


Figure 6.1: Illustration of camera drift. The two objects should line up vertically, however, by moving the microscope, undesired drift has occured.



(a) Overlapping images

(b) Completed stitching of the images

Figure 6.2: Stitching process of overlapping images

6.2 Stitching & segmentation

6.2.1 Approach

A substantial part of the system is the stitching and segmenting of objects found from the acquisition stage. Stitching refers to the process of combining overlapping images into a single, larger image, while segmentation is the process of detecting and isolating objects on that image. The prominent way to approach this issue is with the field of computer vision (CV). CV is a scientific field which is concerned with getting an understanding of the physical world with the use of digital computers[14]. Some sub-domains of CV to consider for stitching and segmentation are deep learning and digital image processing. They both have their respective advantages and disadvantages which have led to some CV implementations using a combination of the technologies[15]. Deep learning is a more modern technology and therefore has extending capabilities. For one, it is better at detecting objects in an image with a noisy or messy background. However, DIP generally has less demanding techniques, which means it typically requires fewer processing resources. Furthermore, it provides great customizability and clarity in the design process. The DFI system has to segment the images from a mostly uniform background. It is therefore positively within the scopes of the capabilities of DIP. DIP is also a more mature technology, which means that it has been tried and tested over time. There exists a vast number of openly available tools and resources which have been optimized for both robustness and fast computations. It was therefore decided that DIP would primarily be used for the requirements specified in use cases 3 and 4, as explained in sectino 5.2.2.

6.2.2 Methods

The approach to image stitching can be summarized as registration and matching of features in multiple, overlapping images and the compositing of these to a final image. This is shown in figure 6.2 where six images with overlapping features can be combined into a single image. The concrete operations for a sufficient stitching operation consists of many, smaller complicated steps. The DFI system will thus rely on existing tools to complete the stitching, as some of these have been greatly optimized for the process.

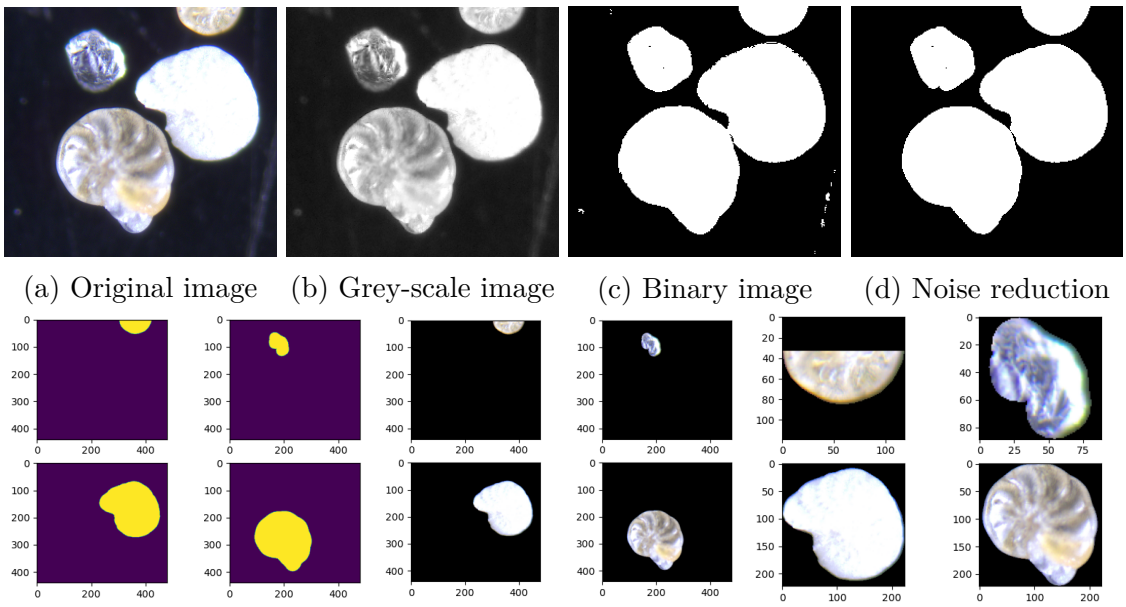
The segmentation process can be broken down into a few discrete steps. These steps follow a conventional approach which is fitting for the specifications of this system[16]. These are shown in figure 6.3. The image is first grayscaled and a binary image is created. Then it is enhanced and has removed noise with morphological operations. The separated

segments are then isolated and mapped back to the original image. Finally, the images are cropped to a consistent format to be passed to the classification step. The execution and theory behind these steps will be explored further in the 'Theory' chapter[17].

6.2.3 Tools

Two of the state-of-the-art tools related to CV are scikit-image and OpenCV, which are both tools developed for Python[18][19]. They are both capable of handling all the common image processing tasks with minor exceptions. Of the two, OpenCV is a more extensive tool[20]. This means that it has some useful applications which can not be handled by scikit-image, but it comes at the cost of time and efficiency as it can be more resource demanding . Therefore, if resources are limited or efficiency is critical, scikit-image is the ideal tool. DFI will therefore rely on scikit-image v0.19.2 where possible. However, for some specific tasks, OpenCV or even more specialized tools might be unavoidable. One of these tasks that scikit-image can not handle, is the image stitching. Even though OpenCV is capable of image stitching, it is not tailored to the specifications of the DFI system. OpenCV does not track relative position, which means that for a given collection of images, it has to try to solve the puzzle of where the images go. This means the use of the computational resources drastically increase, as the collection of images increases.

As the DFI system is designed such that the relative position of the images are known information, a tool which can take advantage of this is ideal. Here a tool like stitch2d has been considered[4]. This tool was designed for stitching images collected from a microscope, where the relative position was known (much like the approach for the DFI system). This means that it has a great performance in comparison to OpenCV as the number of images increase. Unfortunately, stitch2d came with other limiting factors where the stitching would not satisfy the systems requirements. The critical flaw was that even though it was able to place the images correctly, the stitching would not be completed sufficiently. Images would simply be places next to one another, which resulted in the same issue stitching was meant to fix to begin with; the chances of duplication. This



(a) Original image (b) Grey-scale image (c) Binary image (d) Noise reduction

(e) Isolated segments (f) Segments with colour (g) Cropped objects

Figure 6.3: Segmentation with Digital image processing steps

Table 5. **EfficientNet Performance Results on Transfer Learning Datasets**. Our scaled EfficientNet models achieve new state-of-the-art accuracy for 5 out of 8 datasets, with 9.6x fewer parameters on average.

	Comparison to best public-available results						Comparison to best reported results					
	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)
CIFAR-10	NASNet-A	98.0%	85M	EfficientNet-B0	98.1%	4M (21x)	†Gpipe	**99.0%**	556M	EfficientNet-B7	98.9%	64M (8.7x)
CIFAR-100	NASNet-A	87.5%	85M	EfficientNet-B0	88.1%	4M (21x)	Gpipe	91.3%	556M	EfficientNet-B7	**91.7%**	64M (8.7x)
Birdsnap	Inception-v4	81.8%	41M	EfficientNet-B5	82.0%	28M (1.5x)	GPipe	83.6%	556M	EfficientNet-B7	**84.3%**	64M (8.7x)
Stanford Cars	Inception-v4	93.4%	41M	EfficientNet-B3	93.6%	10M (4.1x)	‡DAT	**94.8%**	-	EfficientNet-B7	94.7%	-
Flowers	Inception-v4	98.5%	41M	EfficientNet-B5	98.5%	28M (1.5x)	DAT	97.7%	-	EfficientNet-B7	**98.8%**	-
FGVC Aircraft	Inception-v4	90.9%	41M	EfficientNet-B3	90.7%	10M (4.1x)	DAT	92.9%	-	EfficientNet-B7	**92.9%**	-
Oxford-IIIT Pets	ResNet-152	94.5%	58M	EfficientNet-B4	94.8%	17M (5.6x)	GPipe	**95.9%**	556M	EfficientNet-B6	95.4%	41M (14x)
Food-101	Inception-v4	90.8%	41M	EfficientNet-B4	91.5%	17M (2.4x)	GPipe	93.0%	556M	EfficientNet-B7	**93.0%**	64M (8.7x)
Geo-Mean						**(4.7x)**						**(9.6x)**

Figure 6.4: Performance of EfficientNet compared to other state-of-the-art models [26, table 5].

resulted in the system ultimately being developed using OpenCV v4.5.5 for stitching.

6.3 Classification

When researching existing technologies for image classification, it is important to make use of the fact that an existing dataset exists. A previous bachelor project working on this, FRIDA [3], created the dataset. This makes it possible to use machine learning, specifically supervised learning, to create a model capable of classifying foraminifera from images.

Supervised learning uses classification algorithms and regression techniques to develop predictive models. These models match an input to a finite number of discrete categories. Specifically, this is a classification problem[21]. In the field of image classification in machine learning, convolutional neural network (ConvNet) are currently a widely used method, as it reduces the high dimensionality of images without loosing information [22][23]. For this reason, among others, the choice to use a ConvNet model for classification was made.

However, there exists a plethora of different ConvNet models [24][25], tested on many different existing datasets which are used to set a benchmark for today's models. The previous bachelor projects already explored multiple models. Notably, ResNet50-V2 were the most successful implementation when looking at both size and speed. With ResNet50 it was possible to create even deeper networks by introducing a skip layer. However, a model called EfficientNet[26][27] later appeared, which introduced a new way of scaling networks with a method called compound scaling. Previously, scaling was done manually and was very time consumming; EfficientNet eliminates this, by scaling systematically in width, depth and resolution, based on a what they called the *compound coefficient*, rather than scaling it based on testing. This introduced a more precise model with less parameters. A comparison to other state-of-the-art neural networks can be seen in figure 6.4. With this in mind, it was decided to create and use an EfficientNet model for this project, specifically EfficientNetB7.

6.3.1 Testing of EfficientNet

The classification will only consist of two classes: foraminifera and sand. Since we're working with the FRIDA dataset, this would mean there is three times as many images of foraminifera available, compared to the images of sand. Thus, it would be expected to be more precise in classification of foraminifera.

Confusion Matrix

	precision	recall	f1-score	support
0	0.98	0.99	0.98	91
1	0.97	0.93	0.95	30
accuracy			0.98	121
macro avg	0.97	0.96	0.97	121
weighted avg	0.98	0.98	0.98	121

F1-score: 0.975064239607421

Figure 6.5: Confusion Matrix of the created EfficientNet model

With this, a model was trained. The results can be seen in the confusion matrix in figure 6.5[28].

The figure shows the model have an f1-score[29] of ≈ 97%. The model is 245 MB in size. The model was also tested on the validation data. The average time it took to resize an image to the input size, perform prediction on said image and finally store the image is 0.2719 seconds. All Foraminifera were correctly classified, as expected for reason previously stated. However, four of the 15 images of sand were wrongly classified as foraminifera.

An example of the prediction parameters when classifying an example foraminifera image and sand image is depicted in figure 6.6.

(a) Image of foraminifera (b) Image of sand

Figure 6.6: Showcasing model prediction on a foraminifera and sand. The model predicted [0.9990 0.1105] on 6.6a and [0.2454 0.9853] on 6.6b, meaning it classified both correctly.

As stated in the caption, the model predicted correctly both images, though the prediction numbers show it's more precise when predicting foraminifera than sand. A full description of the developed model will follow in the chapter 8.6. For the scope of this project, the model was deemed acceptable.

6.3.2 Tools

The training of the model will be done in Python, due to the amount of existing libraries. JupyterLab will be used to run the code. For ConvModels, the two most used libraries Pytorch and Tensorflow. The difference between the two can be found online [30][31]. Tensorflow was chosen, as the guides found online for creating a model were developed Tensorflow, while it also seemed more straightforward to use and had better visualization. Tensorflow[32] 2.8.0 with Keras was the one used. The model will be developed following two online guides [33][34].

Chapter 7

Theory

7.1 Acquisition - Physical image theory

A digital microscope works by applying the optics of a traditional microscope and the sensor of a digital camera. An image is obtained in much the same way, as a conventional digital camera, which means similar parameters can be considered for the quality of the final image. Primarily, this can be facilitated as three main topics; the focus of the lens, the time the sensor is exposed to light and the sensor's sensitivity to light.

The first of these parameters to consider is how to keep the image sharp which is greatly affected by the optics of the lens. More specifically, it is controlling if an object is in focus. As the light from the object passes into the lens it is refracted by the lens as shown in figure 7.1.

Here S_1 is the distance between the object and the camera lens, S_2 is the distance between the lens and the image and F is the focal length. Modifying any of these parameters (or the lens shape) will put the image either in focus or out of focus. The object of interest appears the sharpest once the distance S_1,S_2 and the focal length fit together such that the same spot from the real world maps to the same spot on the sensor.

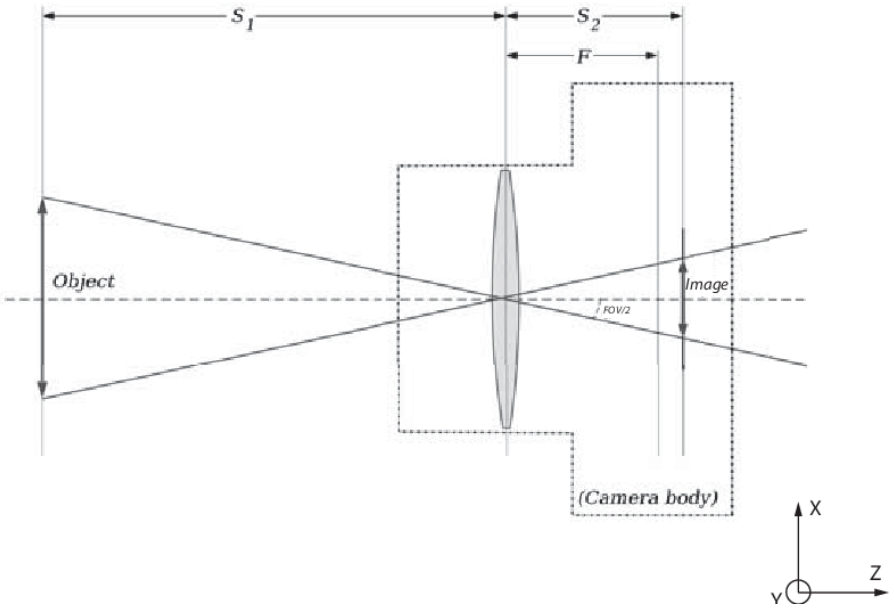


Figure 7.1: Schematic overview of a camera with lens[35, Figure 2]

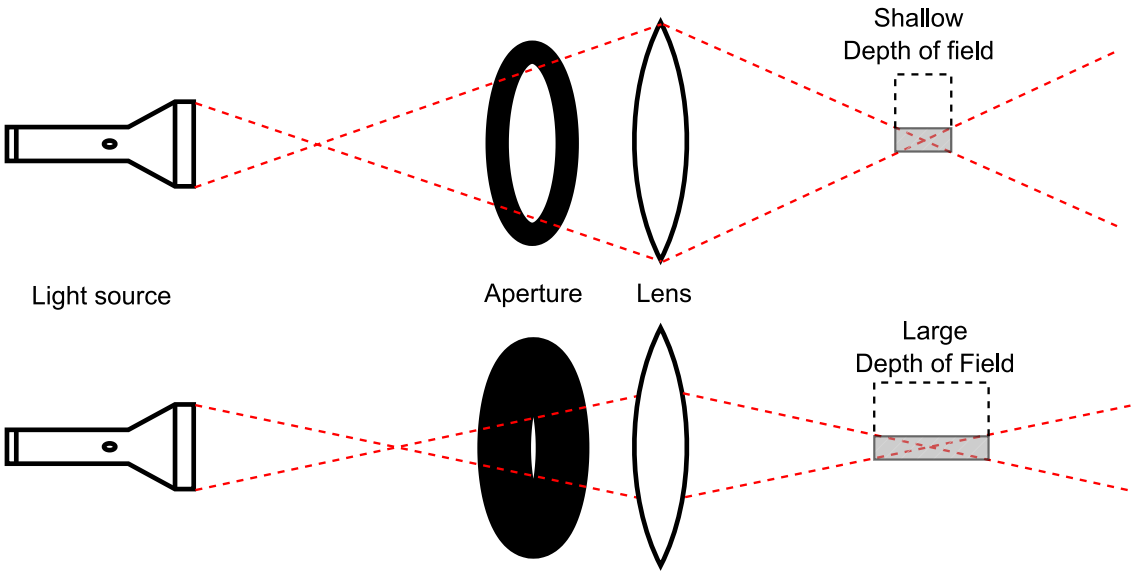


Figure 7.2: Aperature diagram

However, there is an additional parameter that contributes to the sharpness of an image; the aperture. The aperture is the parameter that describes the physical size of the opening of the lens. The aperture controls how the depth of field changes which then determines how far in front and behind the object in focus is also in focus[36]. This is illustrated in figure 7.2 where the depth of field becomes shallower the larger the aperture is. Additionally, these parameters become more and more sensitive the greater the magnification of the image. In turn, this greatly amplifies the effect of any small changes to the parameters. For the implementation of the DFI system, this has the impact that these settings have to be calibrated with quite a large precision. Nevertheless, it is not unlikely that some granular material is too large for the system. Meaning that even with the smallest aperture, the depth of field of the microscope is too shallow resulting in either the top or bottom of the material inevitably getting out of focus.
The final aspect to take into account for the DFI system is the interaction with the camera sensor.

Regarding the sensor there are two primary adjustable parameters; the time the sensor is exposed to light and the sensor's sensitivity to light. The time the sensor is exposed to light is referred to as the exposure time and determines for how long the photon detecting pixels will be reacting to incoming photons. Increasing the exposure time will then in turn collect more information from the image. However, for capturing many images, increasing the exposure time has the downside of being slower as the camera has to pause for longer for each image. Additionally, the image becomes more sensitive to instabilities and vibrations as any movement of the sensor, while exposed, can make points in an image map to different pixels, essentially blurring the image.

Fortunately, these downsides can be counteracted by the final adjustable parameter, the sensor's sensitivity to light. This parameter simply works as an amplifier of the digital value for each sensor pixel. This results in the possibility of getting an image of similar brightness as one with a longer exposure time while mitigating the previously mentioned drawbacks. The issue of increasing the sensitivity, however, is that it will likewise amplify any noise in the image. Having noise in an image is inevitable which is why having a lower sensitivity is usually desirable. It is evident that any of these parameters require adjustment of the others for the desired image quality. Balancing the exposure time and

the sensitivity of the sensor becomes a crucial design decision which has to be made when developing a camera-based system.

It is worth noting that all of the above-mentioned aspects of a digital microscope are way more complex than explored in this chapter. For instance, a single microscope lens can consist of dozens of focusing elements and the sensitivity of the sensor is way more complicated than just being a value multiplier. For a system like the DFI system, the parameters and understanding described at the level above are sufficient for understanding the design decisions and implementations.

7.2 Digital Image Processing

7.2.1 Digital image representation

Typically, a digital image can be understood as a 2-dimensional grid (x, y) where each position represents a discrete pixel. Looking at a grey-scale image this pixel has a value within a certain range. Essentially a value of 0 is a black pixel and a value of either 1 or 255 (depending on the format) is a white pixel. For a coloured picture, a common representation is the RGB colour model. This model has much the same structure as a grey-scale image, except the value of a pixel is represented as an array with a length of three. This array refers to the three colour channels; red, green and blue. For each of the channels, the value now represents how much of the respective colour is shown instead of it just being light or dark. A standard notation of an image could thus be:

$$(x, y, [R, G, B])$$

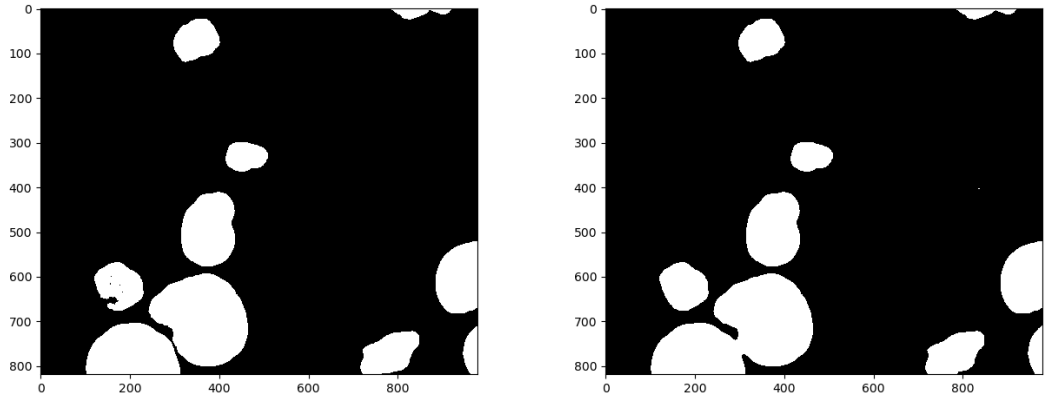
This is one of the more common representations of a digital image. However, there are many other ways to represent a digital image. Each representation has its one advantage and being able to transition between them is the basis of a lot of image processing.

For example, a very useful representation of an image is a binary image. A binary image is shown in figure 7.3c which illustrates that every pixel is either completely black or completely white. The way a binary image is created is first by converting it from a coloured image to a grey-scale image. This is simply done by representing each pixel as the weighted sum of each colour channel for that respective pixel. Converting the image

(a) Coloured image (b) Grey-scale image (c) Binary image

Figure 7.3: Colour representation of image

to a binary image is then done by checking if each pixel is above a certain threshold. For figure 7.3c for instance, each pixel of figure 7.3b with a floating point value less than 0.45 has been turned black and each pixel with a greater value has been turned white.
Binary images are useful for many operations such as morphological operations or segmentation which will be explored later. There are many ways a binary image can be generated



(a) Example of the opening process on an image

(b) Example of the opening process on an image

Figure 7.4: Opening and closing operations applied on the same image

which can highlight various aspects. A useful method is the Otsu method which finds a fitting threshold for a given image[37, p. 752].

7.2.2 Morphological processes

Morphological operations are useful methods to manipulate an image concerning regions and noise. When referring to morphology in digital image processing, it applies mathematical morphology which is a branch of set theory[37, p. 635]. In general, morphology is concerned with the representation and descriptions of shapes and regions of the image.

There are two fundamental morphological operations; erosion and dilation. Erosion and dilation are operations which modify the edges of a foreground element. Simply put, erosion will remove some of the foreground pixels from the edges where dilation will add foreground pixels to the edges. A more detailed description of these operations can be found in the theory appendix[17]. These operations are the basis for many other operations such as opening and closing. Opening and closing are morphological operations where erosion and dilation are used consecutively. Opening is the process of first applying erosion followed by dilation and closing is the reveres process of first applying dilation and then erosion. The effects of these operations are quite apparent from their names. Opening will take a shape and open up any thin areas and closing will fill out any gaps. This can be seen in figure 7.4 where the operations yield different results. The effect is especially apparent on the object in the bottom left. Here the opening process completely splits the two objects where closing merges them more together. Opening is thus the ideal morphological operation in the case where the goal is to split touching objects.

7.2.3 Segmentation

Segmentation is the process of identifying unique elements of an image and isolating them. There are many ways to segment an image depending on both the input and the intended use. One of the more simple methods is the threshold based segmentation[37, p. 742]. This method works by subdividing images directly into objects based on the characteristic values of the pixels. For this reason, binary images are a big part of threshold based segmentation. Here the foreground pixels will be grouped together with neighbouring foreground pixels to generate a set that contains all the pixels of an isolated segment.

7.2.4 Stitching

Image stitching is the process of combining multiple overlapping images into a single image. The general process is to first gather registration data and then composite this data into a larger panoramic image[38]. The registration data is generated with feature extraction by first detecting features and then matching features. Subsets of these features are then used to build a panoramic section. With the registration data generated the compositing process can be executed. Here warping, exposure difference and scaling are evaluated and compensated for such that a final seemliness image can be returned.

7.3 Classification

This section is an extract from the theory documentation and will only include a description of EfficientNet. To see theory on convolutional neural networks (ConvNet), see the documentation[17]

7.3.1 EfficientNet

EfficientNet[26][27] is a specific type of ConvNet, first introduced in 2019. With it, a new method to scale up ConvNets was proposed.

Scaling of models in terms of either width (more Neurons in each layer), depth (more layers), or higher resolution (larger inputs) are ways to improve the performance of a ConvNet. The conventional practice is to arbitrary scale these factors, most commonly by scaling depth by adding more layers, or by width by adding more neurons to certain layers. However, this is typically done by arbitrary, manual tuning without any structure to it.

The creators sought to find a new way to scale ConvNets. In their research they found two key observation, listed below:

1. "Scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models". It's demonstrated in their paper how only increasing one dimension will improve accuracy, but quickly stagnate in performance improvement.

2. "In order to pursue better accuracy and efficiency, it is critical to balance all dimensions of network width, depth, and resolution during ConvNet scaling". With this observation, they saw that a systematic scaling would improve performance more than only scaling one parameter.

With this, they realized it's more optimal to uniformly scale depth, width and resolution, rather than only a single of those. They proposed a method for balancing the scale of all parameters, simply by scaling with a constant ratio. The proposed method was called the *compound scaling method*. The visual difference between the method and other scaling methods is depicted in figure 7.5.

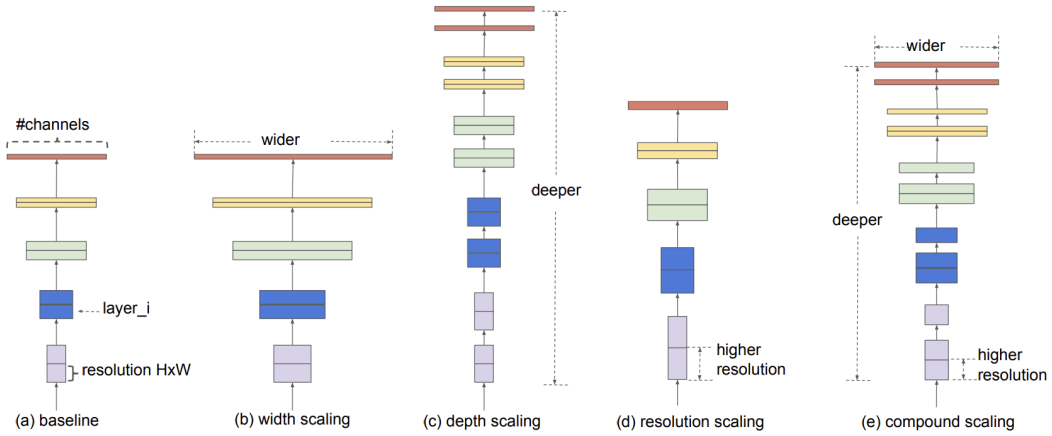


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Figure 7.5: Model scaling as proposed by the EfficientNet developers. [26, Figure 2]

Compound scaling introduces three constant values, α, γ, and β, representing the depth, width, and resolution respectively. Additionally, they introduced ϕ, which is the compound coefficient. The compound coefficient is user-specified, representing how much the model is to be scaled up based on the available computational resources. α, γ, and β specify how to assign those resources. Now, if you wish to use 2^N more computational resources, you scale the size to α^N γ^N and β^N. Importantly, the size of those values are to satisfy the following equation:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

If they do, any new ϕ will ensure the number of FLOPS will increase by $\approx 2^\phi$, for reason argued in the original paper. In order to test this method of scaling, EfficientNetB0 was created. The architecture of the network is shown in figure 7.6. It served as a baseline network and was to be tested upon with the compound scaling method. From it, EfficientNetB1 through B7 were created. The one used in this project will be the B7 version [39], as it's the most precise, though it's also the largest.

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 7.6: Architecture of EfficientNet-B0 baseline network. [26, Table 2]

Chapter 8

Architecture & design

8.1 Introduction

This chapter aims to give insight into the design and architecture of DFI. Additionally, it will describe the implementation that is built upon said designs and architectures. A description of the system is already provided in chapter 3.3, which this is implemented upon, meaning the focus here is on the developed system. This chapter will only present key points in the design and implementation. To see a full description, see the architecture and implementation document[40].

8.2 Deployment diagram

A deployment diagram is a UML diagram, prodiving a high-level view of the architecture and it shows how the software is mapped onto the hardware. The deployment diagram developed for DFI is depicted in figure 8.1. It shows the hardware components and which software is present on them. Each component is explained in more detail below, see table 8.1.

Component type	Specification	Description
Device	PC	The host machine for the OS software to run on.
OS	Windows 11	The OS located on the host device is specified as Windows 11, as DFI has been developed and tested on Windows 11. It should be compatible with previous Windows versions as Windows is generally backwards compatible. However, this has not been tested and thus can't be guaranteed to work.
Software	Pylon 6	An SDK provided by Basler to communicate with and control the digital camera.

Software	Acquisition	The software responsible for use-case 1 and 2: the communication with the camera and the controller. It communicates with the camera via Pylon libraries, which is compatible with C++ in Visual Studio 2013[10]. The communication with the controller goes through a serial port. Here, G-Code[13] commands are sent to the controller.
Software	Stitching and segmentation	The software responsible for use-case 3 and 4: stitching images, and segmentation on said stitched image. It gets access to the images from the Acquisition software through the OS system It is developed in Python.
Software	Classification	The software that is responsible for use-case 5: classifying the objects into their respective classes. It is developed in Python.
CNC machine	Genmitsu 3018 PRO	The CNC machine that the motor and controller is attached to.
Controller	Woodpecker CNC Control Board V3.4	The control board is responsible for controlling the CNC machine. It does this per commands received, in the language G-Code. GRBL 1.1f[12] is installed on the controller which is the brain behind controlling the motors based on said commands.
Motor	Nadalan 42 Stepper Motor	The stepper motors handle the movement in the x and y direction, based on commands sent from the controller.
Camera	Camera system	The whole, fully operational system for grabbing images.
Digital Camera	Basler acA2440-35uc	The digital camera is responsible for grabbing the images and transferring them back to the OS system. Is does this by getting commands via a serial port, from which the embedded Basler software located on the camera handles the grabbing [8].
Lens	M Plan APO 5x	The lens magnifies the sediment samples for the camera, such that they are of a sufficient resolution on the images.

Table 8.1: Details of the deployment diagram's components

As the diagram shows, DFI is intended to be run on a single PC connected to a separate camera system and a separate CNC machine. The camera system is a combination of a lens and a digital camera, which connects to and communicates with the PC through a USB port. The CNC machine consists of a controller, three motors (representing the x, y, and z-axis), and a physical structure. The structure is a maneuverable platform, which is able to move the sample relative to the camera. The controller is responsible for the movement by controlling the motors, based on the G-Code commands sent from the PC.

There is located three different software packages on the PC: one for use-case 1 and 2 (acquisition), one for use-case 3 and 4 (stitching and segmentation), and one for use-case 5 (classification). The purpose of locating all software packages on the same PC is for

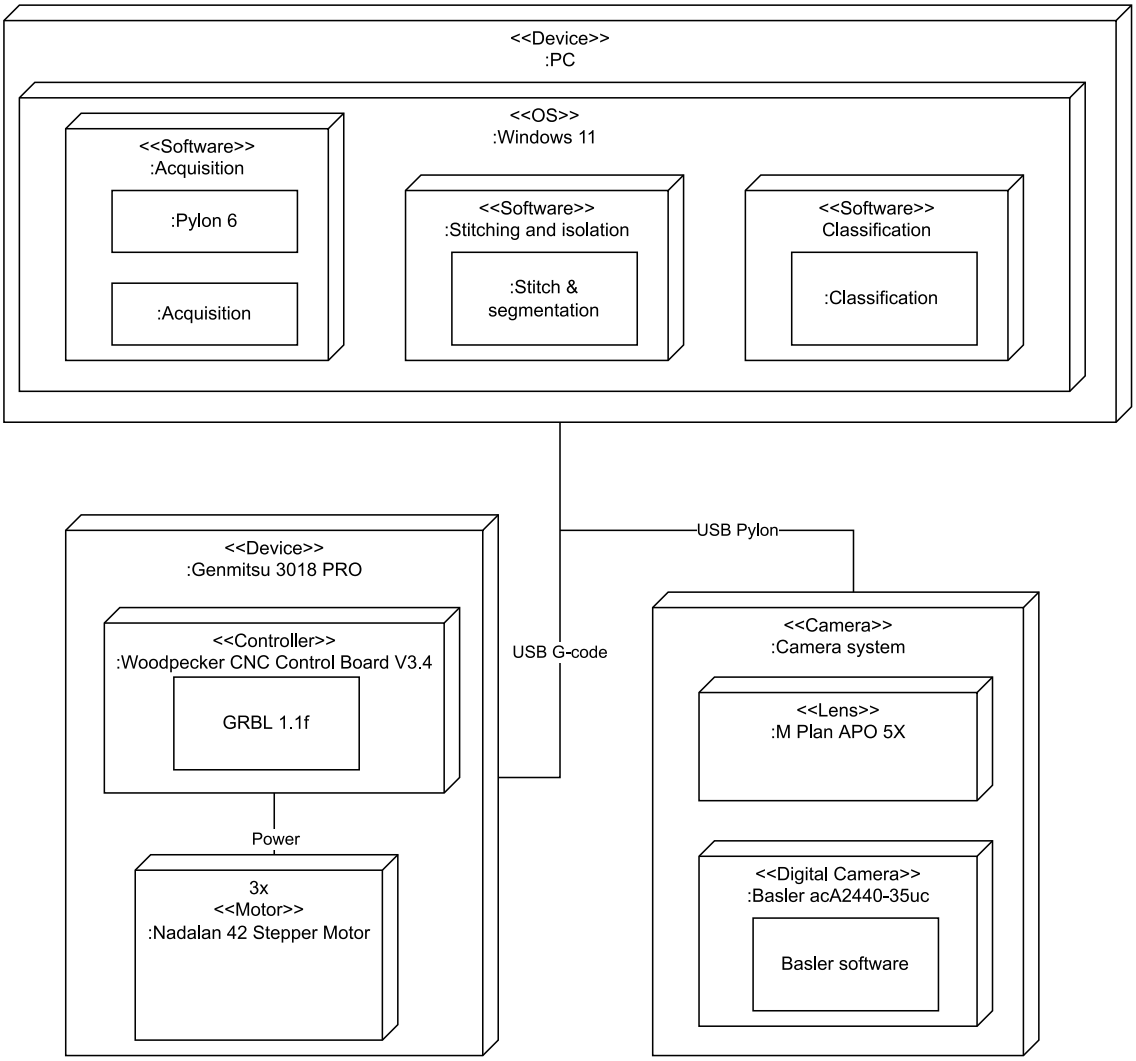


Figure 8.1: Deployment diagram of the full system

easier access to the acquired data and better control flow across the use cases.

8.3 Sequence diagram

A sequence diagram is a UML diagram that shows a high-level view of the dynamic aspects of the system and how processes communicate internally. The sequence diagram developed for DFI is depicted in figure 8.2.

The diagram represents a full walkthrough of all use-cases and is used to show the working of the system. The single actor present is the user of the system (the sediment sample is left out and assumed distributed) and the objects are the software components specified in the deployment diagram, see table 8.1.
First, the user starts the acquisition program, from where they have the option to either set the settings of the camera or start the acquisition. Starting the acquisition leads into a loop that finishes when the whole tray has been covered and there are no sediment samples left to scan. Next, with the acquired images, the stitching and segmentation program is to be started. The program first stitches all the images together and afterward performs

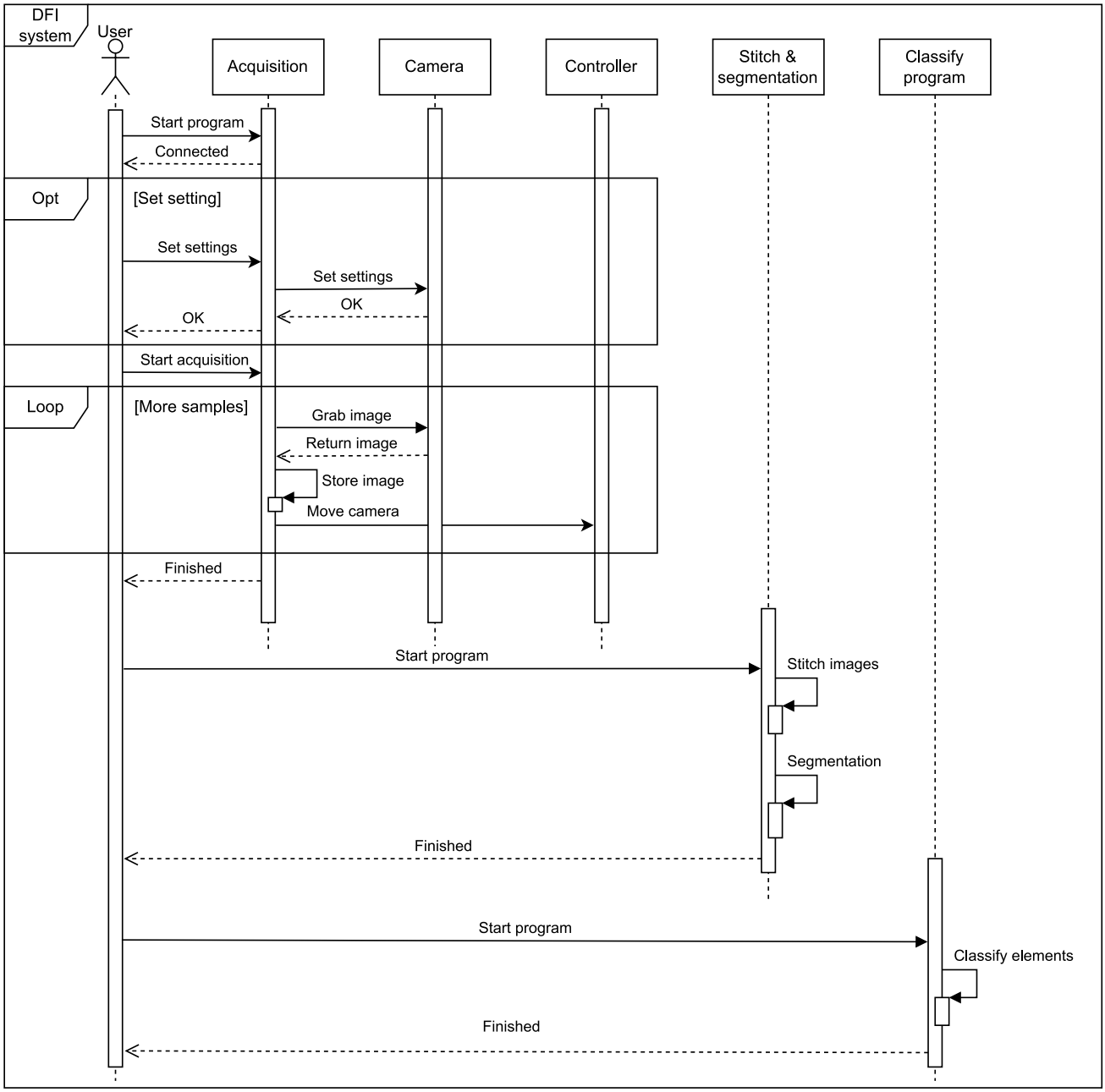


Figure 8.2: Sequence diagram of the full system

segmentation on the stitched image. Finally, with the images of the isolated elements, the classification software is started, which executes the classification and finishes.

8.4 Acquisition

This section will provide an abstract description of the designed acquisition software, responsible for use-case 1 and 2.

8.4.1 Architectural pattern

For reason described in the Architecture & implementation document[40], the acquisition software has been developed following a layered architecture [6, Page 177]. Such an architecture separates the software into different layers, where each layer is dependent on only the layer immediately beneath it. The architecture is depicted in figure 8.3.

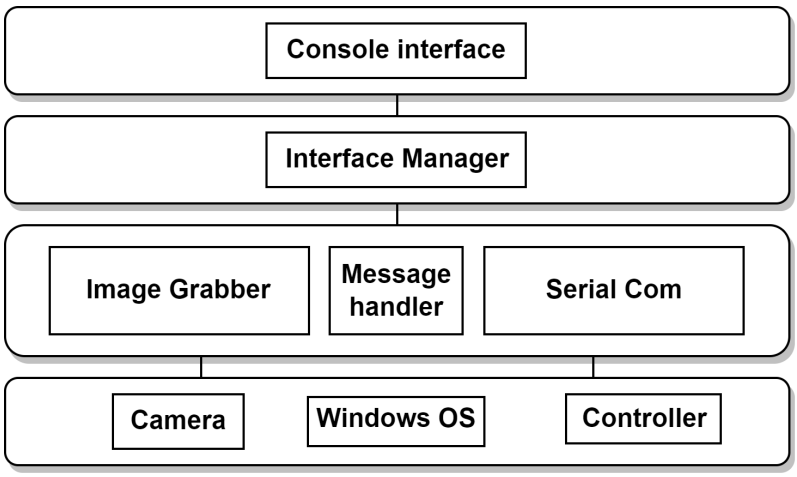


Figure 8.3: The architecture of the acquisition software

The figure shows which elements are present in which layer. In the top layer, you find the simple console interface. The layer beneath it is responsible for handling said interface, based on requests sent from the user. The third layer is the layer responsible for the gathering of data. It is the layer that sends commands to the camera and controller, prompting the movement of the camera and grabbing of images, as explained in the sequence diagram. Finally, the lowest layer is where the data is grabbed and stored in the Windows OS. With this setup a change in one layer will only affect the adjacent layer.

8.4.2 Class diagram

A class diagram have been developed to provide an abstract view. The diagram is depicted in figure 8.4. It contains 4 classes: `ImageGrabber`, `SimpleSerial`, `MessageHandler` and `Interface`. The classes will be described below.

- **Interface:** the class is responsible for managing the user interface of the console. Furthermore, it has an aggregation relationship with the `MessageHandler`, as it has an instance of it as a private attribute `handler`. The instance is the connection to the next layer.
- **MessageHandler:** the class responsible for sending messages to the `ImageGrabber` and the `SimpleSerial`, based on requests received from the `Interface`. It is an aggregation relationship between the `MessageHandler` and the two, as it contains instances of them both as private variables, specifically `grabber` and `serialCom`.
- **ImageGrabber:** the class responsible for the grabbing and storing of images by communicating with the camera software and keeping track of the Windows directory images are stored in.
- **SimpleSerial:** the class responsible for the serial connection with the controller located on the CNC machine. It is based on a free to use class found on Github[41].

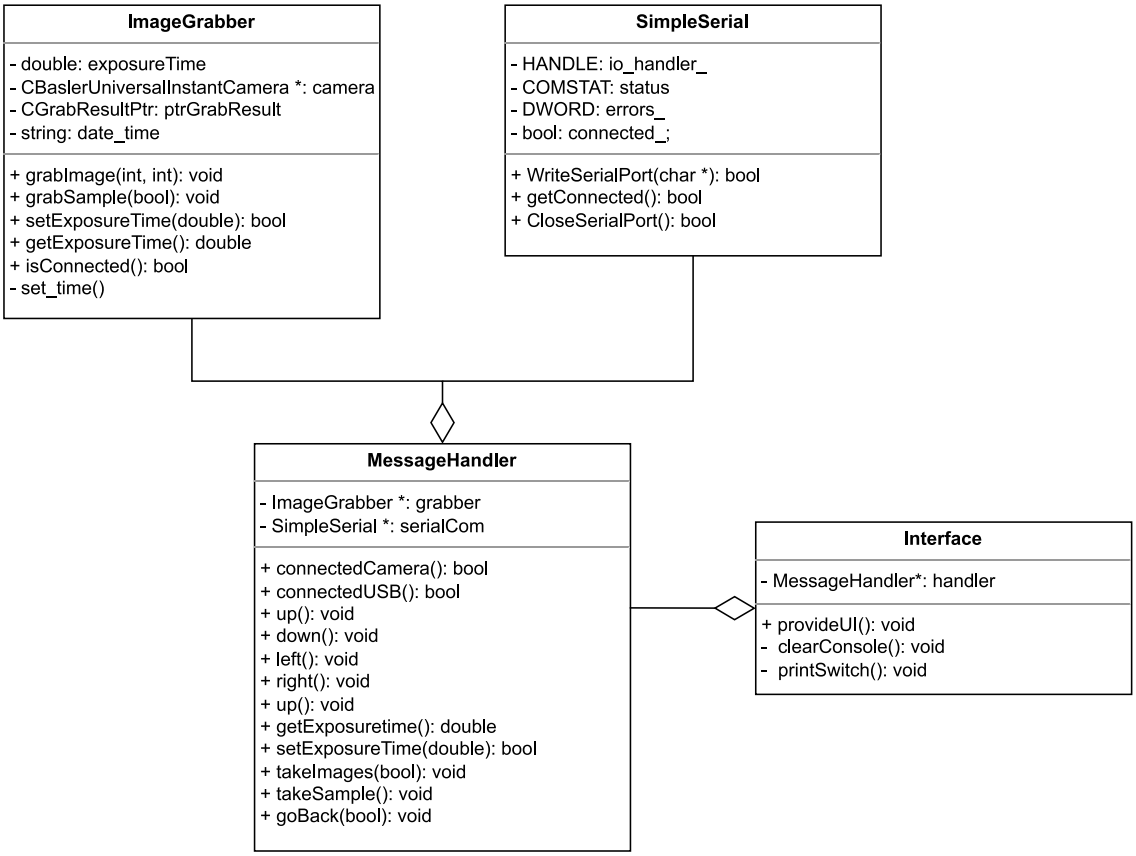


Figure 8.4: Class diagram for the acquisition software

8.4.3 Design choices

Here follows some highlighted design choices made during development. To see them all, see the Architecture & implementation document.

The traversing of the tray is developed with a left-right-left approach, as depicted in figure 8.5.

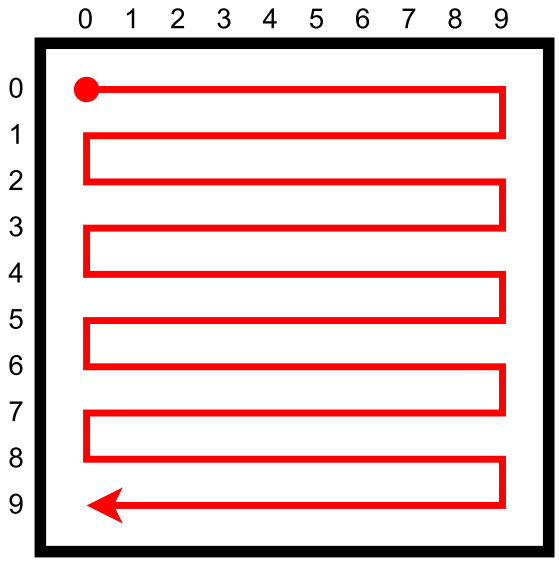


Figure 8.5: The traverse path of the camera.

To ensure a consistent result from the process - without any gaps or drifting - some overlap has been introduced for each image. Too much overlap will both increase the time of the process and require more storage as more images would be captured. Thus, a movement of 1 mm for each image was decided on, as this seemed optimal for each aspect along with it being a simple unit to work with geometrically.

To fulfill FR03 - specified in chapter 5.2 - the motors will stop for 0.5 seconds to allow the camera to grab a clear image. Furthermore, the option to only scan a single quadrant was added, as opposed to the whole tray.

To store the images it was decided not to implement a database. For the scope of this project it was instead decided to use the Windows file explorer, which provides adequate functionality with a user-friendly UI, and provide the ability for easy manual analysis of results.

The naming scheme of the images are designed to represent their position relative to the starting image. Thus, it follows the 'coordinates' like the matrix naming convention of $N \times M$. This means that the top left image is named '0x0.png' and the bottom right is named '9x9.png' when traversing an area of 10x10 mm. This both allows for easier assessment of a scanning run and can provide crucial information in later processing steps. Finally, the images are stored in a directory named the current date-time in the format `'Y-m-d H-M-S'`, which is created when starting the program.

8.5 Stitching and segmentation

This section will provide an abstract description of the designed stitching and segmentation software, responsible for use-case 3 and 4.

8.5.1 Class diagram

A class diagram have been developed to provide an abstract view. The diagram is depicted in figure 8.6. It contains 1 class:

ImageProcessor
The class is created with adequate variables and functions such that it is capable of orchestrating the entire process from a given image collection to the final collection of segmented granular material. The class gets provided a directory for the collection of images to be processed and another directory path to be created, where the segmented objects will be stored. Additionally, it can take a conditional parameter in the form of a complete picture. This is so that if the user has an image of the entire sediment sample, the stitching process can be skipped.

8.5.2 Design choices

As mentioned in the technology research section, image stitching is a resource heavy process. It is thus crucial that the collection of images is optimized beforehand to ensure the best results, especially as the software is intended to run on a PC. The optimization can be done both in the acquisition step or with preprocessing of the images. OpenCV's stitching applies the Brown and Lowe method which is capable of dealing with noise and other inconsistencies itself[42]. This means that not much additional preprocessing is required beforehand, as the method is robust enough to optimize the images itself. Hence, for a given set of images, the only major parameter to consider for optimization is the scale of the images. Downscaling the images to a lower resolution reduces the

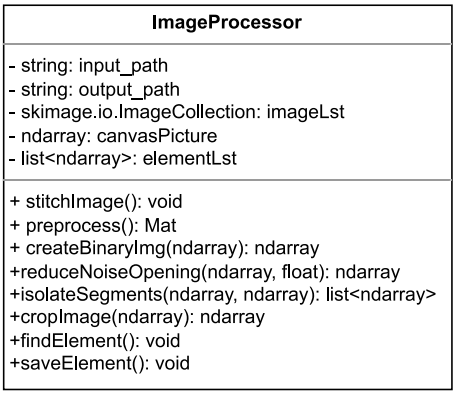
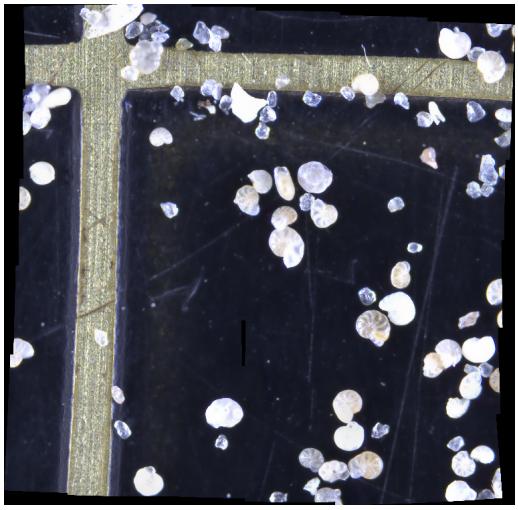
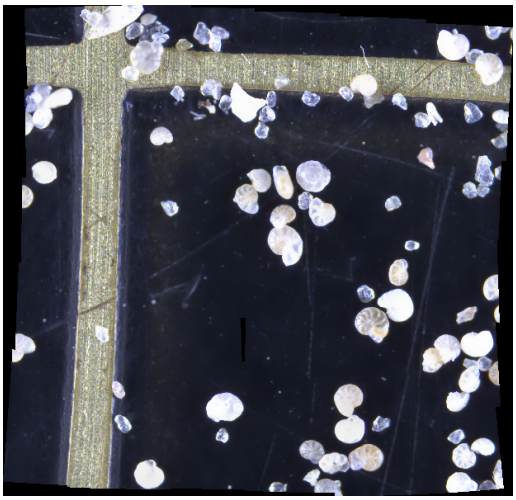


Figure 8.6: Class diagram for the acquisition software

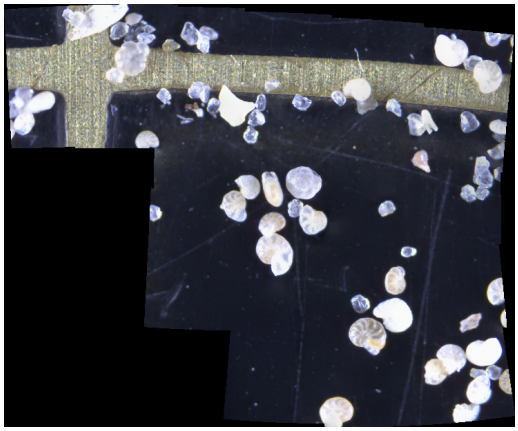
processing demands and cuts down on execution time. It also reduces the chance of the process failing due to running out of memory. However, a lower resolution can remove important information from the images which can cause a worse result or even no result at all. The effect of downscaling the images can be seen in figure 8.7. The figure shows



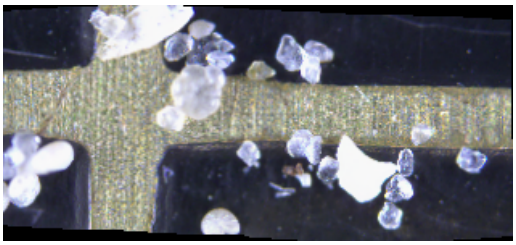
(a) Scale: 1, Avg. runtime: 230.8s



(b) Scale: 0.4, Avg. runtime: 238.1s



(c) Scale: 0.1, Avg. runtime: 10.7s



(d) Scale: 0.07, Avg. runtime: 8.4s

Figure 8.7: Stitching with different scaling of images

the stitching process done of a sample run of a 5x5 mm area consisting of 25 images. Here

it becomes clear that with a lower resolution, particularly for 8.7c and 8.7d, the process cannot detect a connection between more than just a few images and simply leaves them out. A too low resolution, therefore, yields an unacceptable result. In 8.7a and 8.7b, there is no significant difference in runtime as the images are quite similar[1]. As so, the runtime can be completely neglected as a parameter for performance as any stitching less than complete can not be accepted. The desired scale is therefore a scale that will successfully stitch together all of the images but with a scale as low as possible to enable the stitching of a greater number of images without memory error. With the quality and resolution of the images captured in the acquisition stage, it was decided that a downscaling to 60% of the original size was the most optimal value.

For the design of the segmentation software, the first design decision was for the generation of the binary image. The image is first converted to a grayscale image and then to a binary image for a given threshold. This threshold value for foreground pixels can be chosen in many ways but for this system, the choice was either to use the Otsu method or simply decide on a fixed value. The Otsu method can give a different value depending on the provided image. This is useful for images with inconsistent lighting conditions. However, for this system, different acquisition runs - with different amounts and types of granular material - might get a varying threshold. This would in most cases not affect the result as the granular material would likely still be foreground elements. Nevertheless, as the only desired effect is to remove the background, a fixed value will be sufficiently reliable as the images are captured under very controlled conditions with the setup of the system. Furthermore, a constant threshold makes the process more optimized and less complex than running the Otsu process.

8.6 Classification

This section will provide an abstract description of the designed classification software, responsible for use-case 5.

To classify the images, a ConvNet models was trained and exported to .H5 file. With the .H5 file, you can provide inputs to the program, from which it gives a prediction on the type of object. It assigns each possible output class a value in the interval]0;1[, where the value represents a probability of the output being the given class. Naturally, it will be classified as the greatest output value. When an object is classified, the image is stored with the type as its name, either 'Foraminifera.png' or 'Sand.png', in their respective folders.

8.6.1 Design choices

As mentioned in section 8.6, a ConvNet model was trained for the purpose of classification[33]. The model was an EfficientNetB7 model, trained on the FRIDA dataset. The set was 672 images in total, split out across four classes having 168 images each. The decision to train the model on 90 % of the images (608) was made, instead of a standard 80/20 split, as the dataset is quite small. Additionally, it was expected to validate the model on data collected in this project, meaning the size of the validation set could be decreased.

[1]For scales of 0.6 and 0.8, the images were likewise complete and with similar runtime which supports the idea that it is not the scale but the number of preserved images that have the biggest effect on runtime

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 efficientnetb7 (Functional)  (None, 7, 7, 2560)       64097687

 gap (GlobalMaxPooling2D)     (None, 2560)              0

 dropout_out (Dropout)        (None, 2560)              0

 fc_out (Dense)               (None, 2)                 5122

=================================================================
Total params: 64,102,809
Trainable params: 5,122
Non-trainable params: 64,097,687
```

Figure 8.8: Summary of the created model.

A 80/20 split was made on the 608 training images, as an error occurred when trying a 90/10 split for which no working solution was found [43]. The batch size was set to 32, providing an epoch length of 15, and it was trained on 58 epochs. The dropout rate was set to 20 %, while the learning rate was 0.01 %. The input size was 224 x 224, as that was deemed sufficient to classify. The summary of the model can be seen in figure 8.8.

The model is described further in the Architecture and implementation document[40].

It was decided to store all the classified elements as images inside the provided input path. This would be done with a structure like the one seen below:

```
Input path
├── Results
│   ├── Foraminifera
│   │   └── ..
│   └── Sand
│       └── ..
└── ..
```

Here, all images classified as foraminifera are stored in the 'Foraminifera' directory, and all classified as sand in the 'Sand' directory. No GUI was created, and the main function has to be modified to change the input path.

8.6.2 Class diagram

A class diagram have been developed to provide an abstract view. The diagram is depicted in figure 8.6. It contains 1 class responsible for the classification, as training the model is not built as a class. The class present in the diagram is the `Classifier`; a description of the class follows:

Classifier

The classifier is a simple class which takes a path and classifies all `PNG` files inside as either foraminifera or sand. It does so with an .H5 file.

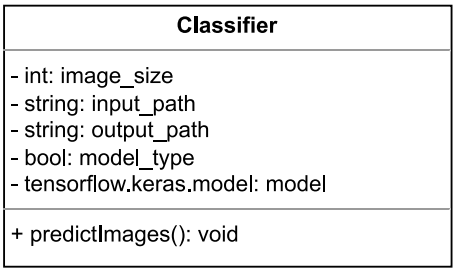


Figure 8.9: Class diagram for the classification software

8.7 Implementation

The specific implementation will not be covered in this document. A thorough description of all functions may be found in the Architecture and implementation document[40]. Along with descriptions of the function, key code snippets are also presented.

Chapter 9

Testing & Results

This chapter will present the results of the test specifications.

9.1 Results of test specifications

The test specifications were defined in the requirements. Below, in table 9.1, is a summary of the accept testing. For a full test on all use cases, please refer to the result documentation[44].

Use case testing summary		
UC No.	**Results**	**Comments**
UC1	(✓)	The only changeable parameter is the exposure time and no live view feed is provided. Instead, it's possible to get a sample image. This considered, it passed with remarks. The presented menu can be seen in figure 9.1.
UC2	✓	It acquires the images as intended. An example of captured images is shown in 9.2a. The results from an acquired quadrant can be found in appendix[45].
UC3	÷	A test on a full acquisition data-set could not be performed, as no available computer has enough processing power or memory to perform the stitching. The amount of time also greatly depends on the number of images. A successful test was performed on 25 images spanning an area of 5x5 mm, where it took about 15 minutes. This stitching is shown in 9.2b. This means it's doable, but it is not yet sufficient to mark it as passed as it will fail if the PC has insufficient processing power or memory.
UC4	✓	
UC5	✓	The classification is not perfect, but above the standard set by the non-functional requirements, meaning it can be marked as passed. A specific classification result can be seen in figure 9.3. Figure 9.3a shows that all isolated foraminifera were correctly classified (though sand19 is a Foraminifera, it is of too low quality to judge based off of it). It also shows 6 out of the 32 isolated sand grains were misclassified, yielding an accuracy of just over 81 percent. Cases where both a sand grain and a foraminifera were present are ignored.

Table 9.1: Summary of use case testing.

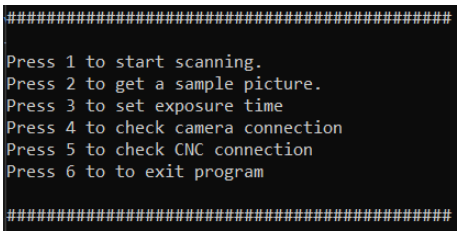
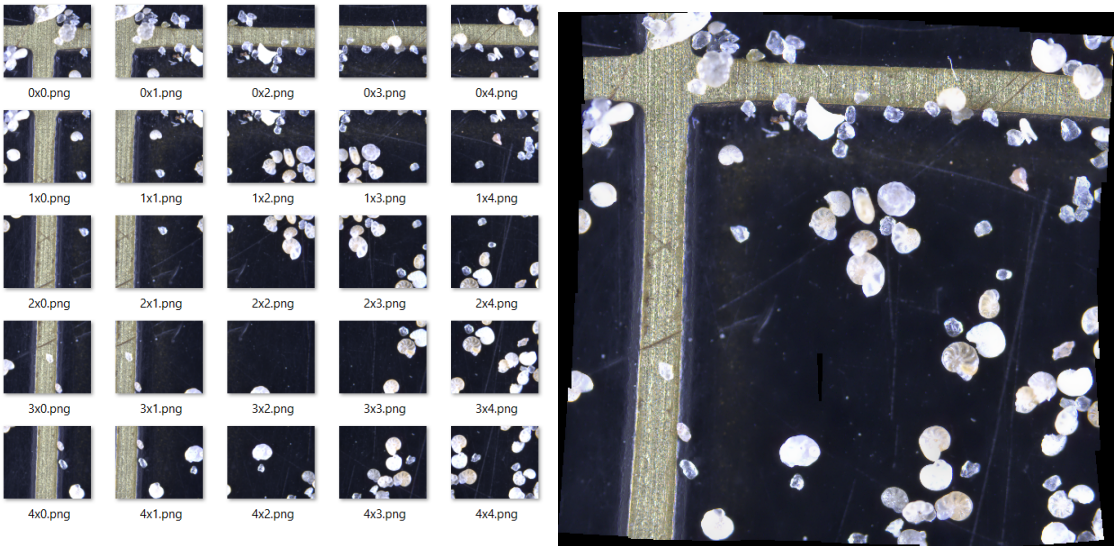


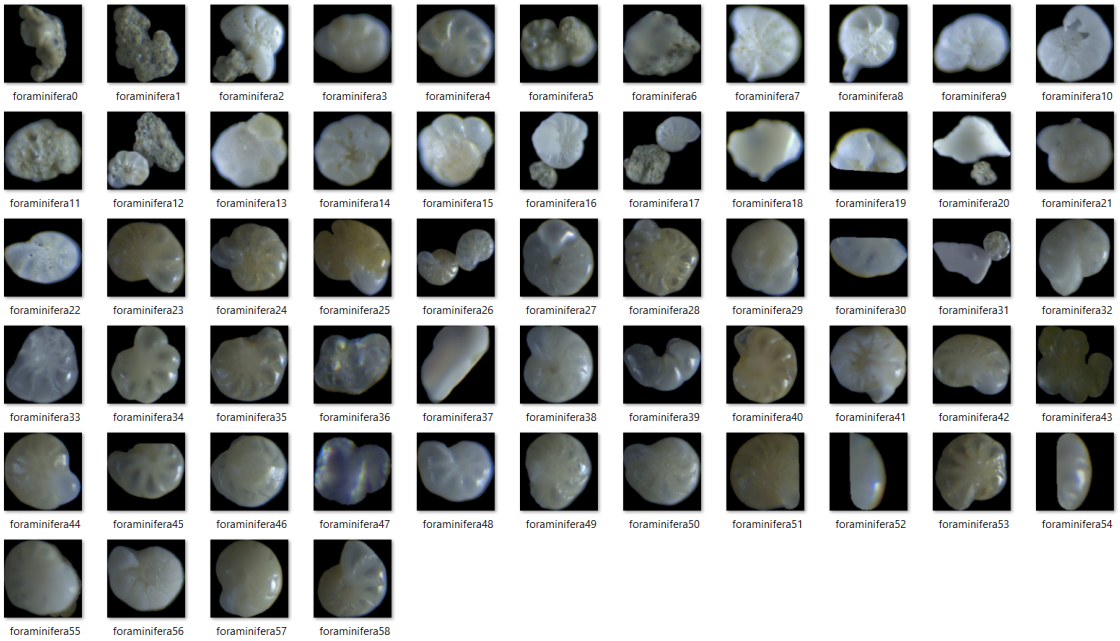
Figure 9.1: The provided interface for setting the acquisition settings.



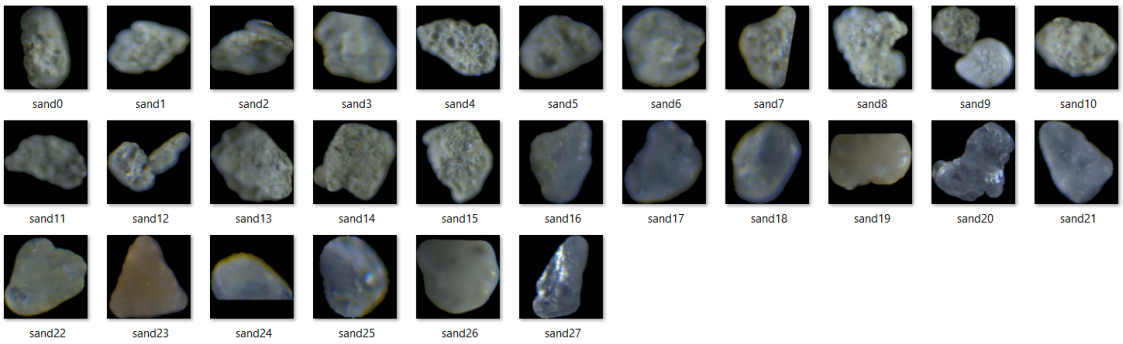
(a) Original images collected from the acquisition stage

(b) Stitched image of the acquired images

Figure 9.2: Illustration of 25 images in a 5x5mm grid stitched together.



(a) Images predicted as Foraminifera.



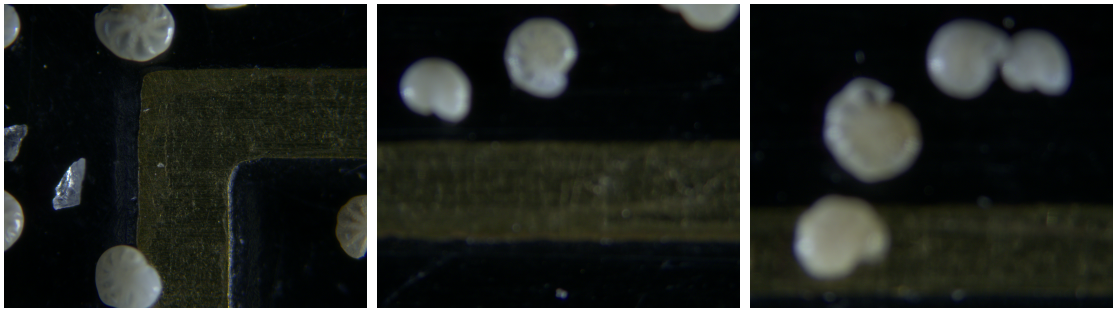
(b) Images classified as sand.

Figure 9.3: Result of the classification program on a sample from segmentation.

9.1.1 Other results

Other notable results gained from testing are listed below:

- A full acquisition provides images around 34,7 GB in size.
- As seen in figure 9.4, images quickly get out of focus when moving along the x-axis.



(a) Image 0x0.png (b) Image 0x35.png (c) Image 0x72.png

Figure 9.4: Result of the classification program on a sample from segmentation, showing how they get gradually more and more out of focus. The naming scheme follows millimeters.

Chapter 10

Discussion

As the design, implementation and testing of the system have concluded, this section will provide a discussion of the results from the testing, notable observations, a reflective look and considerations for changes and future work.

As seen from the results, the DFI system is capable of generating a classification of individual objects from a provided sediment sample. Nevertheless, some aspects and conditions had to be carefully considered in order to successfully fulfil the requirements. The first major complication is the calibration and alignment of the setup. After having executed multiple tests, it has been realized that adjustment of the tray positioning, microscope positioning and microscope focus is a sensitive and unreliable endeavour. This issue can be improved by making some modifications to the setup. One way this might be achieved is by reconstructing the setup, such that the observation area for the sample is fixed to the CNC system, as well as implementing a positional calibration feature, such that the system can identify the position of the microscope. This would allow the system to move the microscope to a consistent starting position, which will remove the issue of manual fine tuning and alignment.

Another complication discovered during acquisition testing is the limitations of the physical setup. Attempting to run acquisition on the entire tray will result in some areas being out of focus, as shown in the figure 9.4, regardless of the initial calibration. This indicates that a change in distance occurs between the microscope and the sample. The issue could be caused by many factors, although it is thought that it is most likely either an error with the rail of the CNC setup, which the microscope is attached to, or an error with the tray. A subtle change in the distance from the microscope to the sample is enough to make the images useless. Further testing is required to determine the cause, but with the system set up as it is currently, it is not reliable for acquiring data of the entire tray.

The only failed requirement of the system is the stitching. As mentioned above, stitching has been successfully executed, but not completely to the specifications of the requirement. Revisiting the stitching is thus necessary for a complete system. The limitations of stitching are believed to be primarily caused due to the hardware components, as the process is intended to be run on a PC. If the use of a PC - with limited processing power and memory - is not changed, the entire stitching approach has to be redesigned. One of the performed acquisition tests on the entire tray acquired more than 34 GB of data. This is way outside the capabilities of a regular PC to process. An alternative solution for stitching could be to implement a method so that the stitching is only done on individual quadrants of the grid. This would be an area of 10x10 mm and consist of 100 images. With optimizations to the stitching process - such as taking advantage of the relative positioning - it is believed that stitching one quadrant of the grid is possible on a PC.

With this method, the program must then sequentially go through all the grid squares one by one. These partial images must then be passed individually to the segmentation step. Thus, a redesign of the `ImageProcessor` class is required.

For the segmentation of elements it was expected that an issue would occur where some unwanted objects would be generated. For one, it was expected that the grid from the tray would appear. Additionally, it would be inevitable that some fragmented granular material would appear, as the provided image for the process would most likely have incomplete material at the edges of the image. The intended approach to fix this was to set an upper and lower thresholds for the sizes of the final object images. If an image was way too small to be a foraminifera or sand it would be sorted out. This would likewise be the case for the upper limit where objects magnitudes larger than the granular material - such as the tray grid - would be removed. However, as the stitching process turned out to be quite extensive, changes to image scaling would be modified quite deep into the development process. This would affect the values for the upper/lower scaling threshold meaning the size filtering was never implemented. If this was known in advance, the `ImageProcessor` class would have been designed with considerations for more sophisticated scaling characteristics. This would allow for a modifiable definition of the granular material size such that filtering too large and too small objects would be possible. Moreover, as foraminifera can vary widely in size, allowing the user to set these thresholds could extend the potential uses of the system to the desire of the user[1].

A more unexpected outcome of the segmentation has been the extend to which overlapping material would be segmented as a single object. It was presumed that some grouping would take place, yet with careful spreading of the sample in the acquisition stage, the grouping was thought not to be of significant issue. However, this was not the case, as grouping proved to be quite significant and troublesome. An entire group of elements is unusable for classification, which in turn means that just a few groupings will critically affect the result. In addition, the grid of the tray protrude slightly from the surface. This further causes grouping as the material tends to clump up around corners of the grid. Fixing this issue could be done by redesigning the way the sample should be distributed on the tray. Manually shaking the tray seems to only worsen the issue as lighter material settles around heavier material. It would, however, be worth exploring if shaking the tray at higher frequencies with a motor would yield a better result. Another possible solution is to handle the overlapping digitally. There exists methods in the field of DIP which can detect spherical objects such that they can be isolated. One approach could be by introducing the Hough transform which is a method of detecting lines and spheres in objects[46].

The granular material touching the grid should likewise be addressed. With the setup as it is now, the image processing is not capable of separating granular material from the grid. It might be possible to solve this issue by the use of DIP with some more advanced filters. However, this process is not trivial, as the light values and colour of the grid is quite similar to the granular material. The simplest and possibly most reliable solution would be to replace the tray with a different one. More specifically, choosing an observation platform where the color of the grid has a bigger contrast to the sediment sample. Switching to a tray with a more distinguishable grid colour, such as red, would make it much simpler to remove the grid with DIP, as the colour channels are easier to separate.

The trained EfficientNet model for the system performed quite well in the tests. However, it can still be optimized by providing it a larger dataset to train on, as the current dataset has more foraminiferas than sand, and is quite small in general. The larger amount of foraminifera samples meant the model was less precise when classifying sand, which is

undesired. Another factor is that the resizing of images in the stitching process meant the segmented samples were often of a size smaller than the input size to the model. This meant that they had to be scaled up, making them of less quality. This in turn would make it harder for the model to classify correctly, which is undesirable. Additionally, for the samples provided for testing, the sand was smaller than the foraminifera.

Moving forward other types of models can be trained, and more design space exploration can be performed to optimize the EfficientNet model. However, expanding the dataset would definitely be optimal before exploring more models. It is possible to use the Endless Forams dataset [47], though gathering data from this system would be better to train on, as it would be more alike the future gathered data in exposure, lighting and image quality.

10.1 Future work

As the purpose of this project has been 'proof-of-concept', there are still many aspects that require further development. Therefore, possible future development will be presented in this section. Many considerations have been discussed, which in turn introduced possible new solutions. Here follows the previous discussed topics, alongside some new ones:

- Redesign stitching as mentioned in the discussion.
- Develop one user interface for all use cases, either on an online platform or a as local program.
- In the segmentation process, filter out objects that are either too large or too small to be foraminifera, such as the grid. Also, don't include fragmented objects.
- Either find a way to physically separate grouped objects, or separate them with DIP methods.
- Create a larger dataset to train the model upon.
- Auto detect serial port connections in the acquisition program, as they are manually set as of right now.
- Improve the consistency in setup of acquisition. This includes fixing the change in distance when moving the camera, but also fixating the tray and creating position calibrations for the camera to align up automatically. This could also be done by having the camera move automatically up or down depending on the focus of the image.
- Implement a database to store the images in, so that the data can be accesses remotely.
- Test more ConvNet models, and explore more design spaces on already implemented models, both on the model from this project and the models from previous projects.

Chapter 11

Conclusion

The purpose of this project was to help further develop a system that could automate the process of analyzing foraminiferal micropaleontological samples. Specifically, it served as a proof-of-concept project for testing if it was possible to start with a tray of sediment samples and end with automatically classified, segmented images of the sample elements.

To achieve this, a system for acquiring images of the whole tray was developed successfully. Furthermore, it was proven that it is indeed possible to stitch said images into one large image, though in testing, only 25 images were successfully stitched together. Afterward, the use of digital image processing techniques made it possible to segment objects on the stitched image into individual images of said objects. Finally, in order to classify these images a CNN model with an average F1-score of 97% was developed. It was tested on the segmented images and it successfully completed the test.

With all this in mind, it can be concluded that the project indeed served as a proof-of-concept, showing that it is possible to complete the sought-out task, though much future work is required to make it a complete system.

Bibliography

[1] K. Wetmore. "Foram facts — an introduction to foraminifera." (2022), [Online]. Available: https://ucmp.berkeley.edu/fosrec/Wetmore.html (visited on 05/24/2022).

[2] A. Kathigamanathan, R. K. H. Thomsen, and J. E. E. Sømod, "Udviklingen af et fuldautomatisk system til artsbestemmelse af foraminiferer ved brug af neurale netværk," Dec. 2020.

[3] E. J. Broeders and E. H. Skov, "Foraminifera research, imaging and detection application," Dec. 2019.

[4] A. Mansur. "Stitch2d." (2022), [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/stitch2d/latest/stitch2d.pdf (visited on 05/26/2022).

[5] R. Marchant, M. Tetard, A. Pratiwi, M. Adebayo, and T. de Garidel-Thoron, "Automated analysis of foraminifera fossil records by image classification using a convolutional neural network," *Journal of Micropalaeontology*, vol. 39, no. 2, pp. 183–202, 2020. DOI: 10.5194/jm-39-183-2020. [Online]. Available: https://doi.org/10.5194/jm-39-183-2020.

[6] I. Sommerville, *Software Engineering, Global Edition*, 10. edition. Pearson Education Limited, 2016, ISBN: 9781292096148. [Online]. Available: https://books.google.dk/books?id=W_LjCwAAQBAJ (visited on 03/29/2022).

[7] L. K. Vindbjerg and D. C. Biørrith. "Requirement & testing documentation." (2022), [Online]. Available: https://github.com/Biorrith/Bachelor-project/tree/main/References.

[8] Basler AG. "Aca2440-35uc - basler ace." (2022), [Online]. Available: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca2440-35uc/ (visited on 06/09/2022).

[9] Mitutoyo Europe GmbH. "M plan apo 5x." (2022), [Online]. Available: https://shop.mitutoyo.eu/web/mitutoyo/en/mitutoyo/05.04.02/Objective%20Lens%20M%20Plan%20APO%205X%205X/$catalogue/mitutoyoData/PR/378-802-6/index.xhtml (visited on 06/09/2022).

[10] Basler AG. "Pylon 6.0.1 camera software suite windows." (2022), [Online]. Available: https://www.baslerweb.com/en/sales-support/downloads/software-downloads/pylon-6-0-1-windows/ (visited on 03/29/2022).

[11] ——, "How to build basler pylon c++ applications with free microsoft visual studio." (2017), [Online]. Available: https://www.baslerweb.com/fp-1508418766/media/downloads/documents/application_notes/AW00064406000_Application_Note_Build_pylon_C_Apps_with_Free_Visual_Studio_IDE.pdf (visited on 03/29/2022).

[12] Grbl CNC controller. "Grbl." (2021), [Online]. Available: https://github.com/grbl/grbl (visited on 04/04/2022).

[13] RepRap. "G-code." (2022), [Online]. Available: https://reprap.org/wiki/G-code (visited on 04/04/2022).

[14] I. Mihajlovic. "Everything you ever wanted to know about computer vision." (2019), [Online]. Available: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e (visited on 06/13/2022).

[15] N. O. Mahony, S. Campbell, A. Carvalho, *et al.*, "Deep learning vs. traditional computer vision," *Advances in Intelligent Systems and Computing*, vol. 943, 128—144, 2020. DOI: 10.1007/978-3-030-17795-9_10. [Online]. Available: https://rdcu.be/cO7PW.

[16] M. Dyrmann and P. Christiansen, *Automated Classification of Seedlings Using Computer Vision*, 1. edition. Aarhus University, 2014, ISBN: 978-87-997533-0-7. [Online]. Available: https://portal.findresearcher.sdu.dk/en/publications/automated-classification-of-seedlings-using-computer-vision-patte (visited on 06/12/2022).

[17] L. K. Vindbjerg and D. C. Biørrith. "Theory documentation." (2022), [Online]. Available: https://github.com/Biorrith/Bachelor-project/tree/main/References.

[18] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, *et al.*, "Scikit-image: Image processing in python," *PeerJ*, vol. 453, no. 2, 2014. DOI: /10.7717/peerj.453. [Online]. Available: https://peerj.com/articles/453/.

[19] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000. [Online]. Available: https://opencv.org/ (visited on 06/09/2022).

[20] A. Jamshed. "Opencv vs skimage for image analysis. which one is better?" (2021), [Online]. Available: https://medium.com/analytics-vidhya/opencv-vs-skimage-for-image-analysis-which-one-is-better-e2bec8d1954f (visited on 06/09/2022).

[21] C. M. Bishop, *Pattern Recognition and Machine Learning*, 2. edition. Springer New York, 2006, ISBN: : 978-1-4939-3843-8. [Online]. Available: https://link.springer.com/book/9780387310732 (visited on 06/10/2022).

[22] N. Lang. "Using convolutional neural network for image classification." (2021), [Online]. Available: https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4 (visited on 06/12/2022).

[23] A. Bonner. "The complete beginner's guide to deep learning: Convolutional neural networks and image classification." (2019), [Online]. Available: https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb (visited on 06/12/2022).

[24] Papers with Code. "Image classification on imagenet." (2022), [Online]. Available: https://paperswithcode.com/sota/image-classification-on-imagenet (visited on 06/12/2022).

[25] A. Shrivastav. "Different types of cnn models." (2020), [Online]. Available: https://iq.opengenus.org/different-types-of-cnn-models/ (visited on 06/12/2022).

[26] M. Tan, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1905.11946v5 (visited on 06/13/2022).

[27] Papers with Code, "Efficientnet," [Online]. Available: https://paperswithcode.com/method/efficientnet (visited on 06/13/2022).

[28] "Confusion matrix." (2022), [Online]. Available: https://en.wikipedia.org/wiki/Confusion_matrix (visited on 06/15/2022).

[29] J. Korstanje, "The f1 score," 2022. [Online]. Available: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6 (visited on 06/14/2022).

[30] K. Dubovikov, "Pytorch vs tensorflow — spotting the difference," 2017. [Online]. Available: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b (visited on 06/14/2022).

[31] R. O'Connor, "Pytorch vs tensorflow in 2022," 2022. [Online]. Available: https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022 (visited on 06/14/2022).

[32] "Tensorflow." (2022), [Online]. Available: https://www.tensorflow.org/ (visited on 06/15/2022).

[33] M. Ibrahim, "An in-depth efficientnet tutorial using tensorflow — how to use efficientnet on a custom dataset," 2021. [Online]. Available: https://towardsdatascience.com/an-in-depth-efficientnet-tutorial-using-tensorflow-how-to-use-efficientnet-on-a-custom-dataset-1cab0997f65c (visited on 06/14/2022).

[34] V. J., "Tutorial on using keras flow_from_directory and generators," 2018. [Online]. Available: https://vijayabhaskar96.medium.com/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720 (visited on 06/14/2022).

[35] B. Ribbens, "Development and validation of a time domain fringe pattern analysis technique for the measurement of object shape and deformation," Ph.D. dissertation, Mar. 2015. [Online]. Available: https://www.researchgate.net/publication/273652708_Development_and_validation_of_a_time_domain_fringe_pattern_analysis_technique_for_the_measurement_of_object_shape_and_deformation.

[36] K. R. Spring and M. W. Davidson. "Depth of field and depth of focus." (2022), [Online]. Available: https://www.microscopyu.com/microscopy-basics/depth-of-field-and-depth-of-focus (visited on 06/03/2022).

[37] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4. edition. Pearson Education Limited, 2018, ISBN: 9780133356779. [Online]. Available: https://www.pearson.com/us/higher-education/program/Gonzalez-Digital-Image-Processing-4th-Edition/PGM241219.html?tab=resources (visited on 03/29/2022).

[38] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International Journal of Computer Vision*, vol. 74, no. 59-73, 2007. DOI: /10.1007/s11263-006-0002-3. [Online]. Available: https://rdcu.be/cPzwI (visited on 06/13/2022).

[39] V. Agarwal, "Complete architectural details of all efficientnet models," 2020. [Online]. Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142 (visited on 06/14/2022).

[40] L. K. Vindbjerg and D. C. Biørrith. "Architecture, design and implementation document." (2022), [Online]. Available: https://github.com/Biorrith/Bachelor-project/tree/main/References.

[41] D. David Michalik. "Simple serial communication between a windows c++ application and an arduino." (2022), [Online]. Available: https://github.com/dmicha16/simple_serial_port (visited on 06/09/2022).

[42] A. Rosebrock. "Image stitching with opencv and python." (2018), [Online]. Available: https://pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/ (visited on 06/10/2022).

[43] "Warning:tensorflow:your input ran out of data; interrupting training." (), [Online]. Available: https://github.com/fizyr/keras-retinanet/issues/1449 (visited on 06/12/2022).

[44] L. K. Vindbjerg and D. C. Biørrith. "Testing results documentation." (2022), [Online]. Available: https://github.com/Biorrith/Bachelor-project/tree/main/References.

[45] ——, "Images of quadrant found in appendix," 2022.

[46] D. G. Bailey, "Segmentation of touching objects," *Image Analysis Unit, Massey University, Palmerston North*, 2007. [Online]. Available: https://www-ist.massey.ac.nz/dbailey/sprg/pdfs/1992_NZIPW_195.pdf (visited on 06/13/2022).

[47] "Endless forams data portal." (2019), [Online]. Available: http://endlessforams.org/ (visited on 06/14/2022).

[48] Genmitsu Industries LLC. "Original genmitsu 3018-pro." (2022), [Online]. Available: https://genmitsu.com/ (visited on 03/29/2022).

[49] Basler AG. "Pylon sdk samples manual." (2021), [Online]. Available: https://www.baslerweb.com/fp-1615186793/media/downloads/documents/users_manuals/AW00148804000_pylon_SDK_Samples_Manual.pdf (visited on 03/29/2022).

[50] Desktop CNC Supply. "Woodpecker cnc control board v3.4." (2022), [Online]. Available: https://www.3018nc.com/product/3018-pro-cnc-router-control-board/ (visited on 03/29/2022).

[51] Edmund Optics Inc. "Imaging electronics 101: Understanding camera sensors for machine vision applications." (2022), [Online]. Available: https://www.edmundoptics.eu/knowledge-center/application-notes/imaging/understanding-camera-sensors-for-machine-vision-applications/ (visited on 06/05/2022).

[52] D. C. Biørrith and L. K. Vindbjerg. "Github repository - bachelor project." (2022), [Online]. Available: https://github.com/Biorrith/Bachelor-project (visited on 06/07/2022).

[53] "Unified modelling language." (2022), [Online]. Available: https://www.uml-diagrams.org/ (visited on 06/09/2022).

[54] Wikipedia foundation. "Windows file explorer." (2022), [Online]. Available: https://en.wikipedia.org/wiki/Windows_File_Explorer (visited on 06/10/2022).

[55] Papers with Code. "State of the art image classification." (), [Online]. Available: https://paperswithcode.com/task/image-classification (visited on 06/13/2022).

[56] A. Thakur, "Relu vs. sigmoid function in deep neural networks," 2022. [Online]. Available: https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks-Why-ReLU-is-so-Prevalent--VmlldzoyMDk0MzI (visited on 06/12/2022).